

man pages section 3: Multimedia Library Functions

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf disposition de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, breveter, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Si ce logiciel, ou la documentation qui l'accompagne, est concédé sous licence au Gouvernement des Etats-Unis, ou à toute entité qui délivre la licence de ce logiciel ou l'utilise pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer des dommages corporels. Si vous utilisez ce logiciel ou matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour ce type d'applications.

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Xeon sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Opteron, le logo AMD et le logo AMD Opteron sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée d'The Open Group.

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation.

Contents

Preface31

Multimedia Library Functions - Part 1 35

 mlib_free(3MLIB) 36

 mlib_GraphicsBoundaryFill_8(3MLIB) 37

 mlib_GraphicsDrawArc(3MLIB) 38

 mlib_GraphicsDrawCircle(3MLIB) 40

 mlib_GraphicsDrawEllipse(3MLIB) 42

 mlib_GraphicsDrawLine(3MLIB) 44

 mlib_GraphicsDrawLineFanSet(3MLIB) 49

 mlib_GraphicsDrawLineSet(3MLIB) 54

 mlib_GraphicsDrawLineStripSet(3MLIB) 59

 mlib_GraphicsDrawPoint(3MLIB) 64

 mlib_GraphicsDrawPointSet(3MLIB) 66

 mlib_GraphicsDrawPolygon(3MLIB) 68

 mlib_GraphicsDrawPolyline(3MLIB) 73

 mlib_GraphicsDrawRectangle(3MLIB) 78

 mlib_GraphicsDrawTriangle(3MLIB) 80

 mlib_GraphicsDrawTriangleFanSet(3MLIB) 85

 mlib_GraphicsDrawTriangleSet(3MLIB) 90

 mlib_GraphicsDrawTriangleStripSet(3MLIB) 95

 mlib_GraphicsFillArc(3MLIB) 100

 mlib_GraphicsFillCircle(3MLIB) 102

 mlib_GraphicsFillEllipse(3MLIB) 104

 mlib_GraphicsFillPolygon(3MLIB) 106

Multimedia Library Functions - Part 2	111
mllib_GraphicsFillRectangle(3MLIB)	112
mllib_GraphicsFillTriangle(3MLIB)	114
mllib_GraphicsFillTriangleFanSet(3MLIB)	119
mllib_GraphicsFillTriangleSet(3MLIB)	124
mllib_GraphicsFillTriangleStripSet(3MLIB)	129
mllib_GraphicsFloodFill_8(3MLIB)	134
mllib_ImageAbs(3MLIB)	135
mllib_ImageAbs_Fp(3MLIB)	136
mllib_ImageAbs_Fp_Inp(3MLIB)	137
mllib_ImageAbs_Inp(3MLIB)	138
mllib_ImageAdd(3MLIB)	139
mllib_ImageAdd_Fp(3MLIB)	140
mllib_ImageAdd_Fp_Inp(3MLIB)	141
mllib_ImageAdd_Inp(3MLIB)	142
mllib_ImageAffine(3MLIB)	143
mllib_ImageAffine_Fp(3MLIB)	145
mllib_ImageAffineIndex(3MLIB)	147
mllib_ImageAffineTable(3MLIB)	149
mllib_ImageAffineTable_Fp(3MLIB)	151
mllib_ImageAffineTransform(3MLIB)	153
mllib_ImageAffineTransform_Fp(3MLIB)	155
mllib_ImageAffineTransformIndex(3MLIB)	157
mllib_ImageAnd(3MLIB)	159
mllib_ImageAnd_Inp(3MLIB)	160
mllib_ImageAndNot1_Inp(3MLIB)	161
mllib_ImageAndNot2_Inp(3MLIB)	162
mllib_ImageAndNot(3MLIB)	163
mllib_ImageAutoCorrel(3MLIB)	164
mllib_ImageAutoCorrel_Fp(3MLIB)	165
mllib_ImageAve(3MLIB)	166
mllib_ImageAve_Fp(3MLIB)	167
mllib_ImageAve_Fp_Inp(3MLIB)	168
mllib_ImageAve_Inp(3MLIB)	169
mllib_ImageBlend1_Fp_Inp(3MLIB)	170
mllib_ImageBlend1_Inp(3MLIB)	171

mllib_ImageBlend2_Fp_Inp(3MLIB)	172
mllib_ImageBlend2_Inp(3MLIB)	173
mllib_ImageBlend(3MLIB)	174
mllib_ImageBlend_BSRC1_BSRC2(3MLIB)	175
mllib_ImageBlend_BSRC1_BSRC2_Inp(3MLIB)	178
mllib_ImageBlendColor(3MLIB)	181
mllib_ImageBlendColor_Fp(3MLIB)	183
mllib_ImageBlendColor_Fp_Inp(3MLIB)	185
mllib_ImageBlendColor_Inp(3MLIB)	186
mllib_ImageBlend_Fp(3MLIB)	188
mllib_ImageBlendMulti(3MLIB)	189
mllib_ImageBlendMulti_Fp(3MLIB)	191
mllib_ImageBlendRGBA2ARGB(3MLIB)	193
mllib_ImageBlendRGBA2BGRA(3MLIB)	194
mllib_ImageChannelCopy(3MLIB)	195
mllib_ImageChannelExtract(3MLIB)	196
mllib_ImageChannelInsert(3MLIB)	197
mllib_ImageChannelMerge(3MLIB)	198
mllib_ImageChannelSplit(3MLIB)	199
mllib_ImageClear(3MLIB)	200
mllib_ImageClearEdge(3MLIB)	201
mllib_ImageClearEdge_Fp(3MLIB)	202
mllib_ImageClear_Fp(3MLIB)	203
mllib_ImageColorConvert1(3MLIB)	204
mllib_ImageColorConvert1_Fp(3MLIB)	205
mllib_ImageColorConvert2(3MLIB)	206
mllib_ImageColorConvert2_Fp(3MLIB)	207
mllib_ImageColorDitherFree(3MLIB)	208
mllib_ImageColorDitherInit(3MLIB)	209
mllib_ImageColorErrorDiffusion3x3(3MLIB)	212
mllib_ImageColorErrorDiffusionMxN(3MLIB)	213
mllib_ImageColorHSL2RGB(3MLIB)	215
mllib_ImageColorHSL2RGB_Fp(3MLIB)	217
mllib_ImageColorHSV2RGB(3MLIB)	219
mllib_ImageColorHSV2RGB_Fp(3MLIB)	221
mllib_ImageColorOrderedDither8x8(3MLIB)	223

mllib_ImageColorOrderedDitherMxN(3MLIB)	224
mllib_ImageColorRGB2CIEMono(3MLIB)	226
mllib_ImageColorRGB2CIEMono_Fp(3MLIB)	227
mllib_ImageColorRGB2HSL(3MLIB)	228
mllib_ImageColorRGB2HSL_Fp(3MLIB)	230
mllib_ImageColorRGB2HSV(3MLIB)	232
mllib_ImageColorRGB2HSV_Fp(3MLIB)	234
mllib_ImageColorRGB2Mono(3MLIB)	236
mllib_ImageColorRGB2Mono_Fp(3MLIB)	237
mllib_ImageColorRGB2XYZ(3MLIB)	238
mllib_ImageColorRGB2XYZ_Fp(3MLIB)	239
mllib_ImageColorRGB2YCC(3MLIB)	240
mllib_ImageColorRGB2YCC_Fp(3MLIB)	241
mllib_ImageColorTrue2Index(3MLIB)	242
mllib_ImageColorTrue2IndexFree(3MLIB)	243
mllib_ImageColorTrue2IndexInit(3MLIB)	244
mllib_ImageColorXYZ2RGB(3MLIB)	246
mllib_ImageColorXYZ2RGB_Fp(3MLIB)	247
mllib_ImageColorYCC2RGB(3MLIB)	248
mllib_ImageColorYCC2RGB_Fp(3MLIB)	249
mllib_ImageComposite(3MLIB)	250
mllib_ImageComposite_Inp(3MLIB)	253
mllib_ImageConstAdd(3MLIB)	256
mllib_ImageConstAdd_Fp(3MLIB)	257
mllib_ImageConstAdd_Fp_Inp(3MLIB)	258
mllib_ImageConstAdd_Inp(3MLIB)	259
mllib_ImageConstAnd(3MLIB)	260
mllib_ImageConstAnd_Inp(3MLIB)	261
mllib_ImageConstAndNot(3MLIB)	262
mllib_ImageConstAndNot_Inp(3MLIB)	263
mllib_ImageConstDiv(3MLIB)	264
mllib_ImageConstDiv_Fp(3MLIB)	265
mllib_ImageConstDiv_Fp_Inp(3MLIB)	266
mllib_ImageConstDiv_Inp(3MLIB)	267
mllib_ImageConstDivShift(3MLIB)	268
mllib_ImageConstDivShift_Inp(3MLIB)	269

mllib_ImageConstMul(3MLIB)	270
mllib_ImageConstMul_Fp(3MLIB)	271
mllib_ImageConstMul_Fp_Inp(3MLIB)	272
mllib_ImageConstMul_Inp(3MLIB)	273
mllib_ImageConstMulShift(3MLIB)	274
mllib_ImageConstMulShift_Inp(3MLIB)	275
mllib_ImageConstNotAnd(3MLIB)	276
mllib_ImageConstNotAnd_Inp(3MLIB)	277
mllib_ImageConstNotOr(3MLIB)	278
mllib_ImageConstNotOr_Inp(3MLIB)	279
mllib_ImageConstNotXor(3MLIB)	280
mllib_ImageConstNotXor_Inp(3MLIB)	281
mllib_ImageConstOr(3MLIB)	282
mllib_ImageConstOr_Inp(3MLIB)	283
mllib_ImageConstOrNot(3MLIB)	284
mllib_ImageConstOrNot_Inp(3MLIB)	285
mllib_ImageConstSub(3MLIB)	286
mllib_ImageConstSub_Fp(3MLIB)	287
mllib_ImageConstSub_Fp_Inp(3MLIB)	288
mllib_ImageConstSub_Inp(3MLIB)	289
mllib_ImageConstXor(3MLIB)	290
mllib_ImageConstXor_Inp(3MLIB)	291
mllib_ImageConv2x2(3MLIB)	292
mllib_ImageConv2x2_Fp(3MLIB)	294
mllib_ImageConv2x2Index(3MLIB)	296
mllib_ImageConv3x3(3MLIB)	298
mllib_ImageConv3x3_Fp(3MLIB)	300
mllib_ImageConv3x3Index(3MLIB)	302
mllib_ImageConv4x4(3MLIB)	304
mllib_ImageConv4x4_Fp(3MLIB)	306
mllib_ImageConv4x4Index(3MLIB)	308
mllib_ImageConv5x5(3MLIB)	310
mllib_ImageConv5x5_Fp(3MLIB)	312
mllib_ImageConv5x5Index(3MLIB)	314
mllib_ImageConv7x7(3MLIB)	316
mllib_ImageConv7x7_Fp(3MLIB)	318

mllib_ImageConv7x7Index(3MLIB)	320
mllib_ImageConvKernelConvert(3MLIB)	322
mllib_ImageConvMxN(3MLIB)	324
mllib_ImageConvMxN_Fp(3MLIB)	326
mllib_ImageConvMxNIndex(3MLIB)	328
mllib_ImageConvolveMxN(3MLIB)	330
mllib_ImageConvolveMxN_Fp(3MLIB)	332
mllib_ImageCopy(3MLIB)	334
mllib_ImageCopyArea(3MLIB)	335
mllib_ImageCopyMask(3MLIB)	336
mllib_ImageCopyMask_Fp(3MLIB)	337
mllib_ImageCopySubimage(3MLIB)	338
mllib_ImageCreate(3MLIB)	339
mllib_ImageCreateStruct(3MLIB)	340
mllib_ImageCreateSubimage(3MLIB)	341
mllib_ImageCrossCorrel(3MLIB)	342
mllib_ImageCrossCorrel_Fp(3MLIB)	343
mllib_ImageDataTypeConvert(3MLIB)	344
mllib_ImageDelete(3MLIB)	347
mllib_ImageDilate4(3MLIB)	348
mllib_ImageDilate4_Fp(3MLIB)	349
mllib_ImageDilate8(3MLIB)	350
mllib_ImageDilate8_Fp(3MLIB)	351
mllib_ImageDiv1_Fp_Inp(3MLIB)	352
mllib_ImageDiv2_Fp_Inp(3MLIB)	353
mllib_ImageDivAlpha(3MLIB)	354
mllib_ImageDivAlpha_Fp(3MLIB)	356
mllib_ImageDivAlpha_Fp_Inp(3MLIB)	357
mllib_ImageDivAlpha_Inp(3MLIB)	358
mllib_ImageDivConstShift(3MLIB)	360
mllib_ImageDivConstShift_Inp(3MLIB)	361
mllib_ImageDiv_Fp(3MLIB)	362
mllib_ImageDivShift1_Inp(3MLIB)	363

Multimedia Library Functions - Part 3	365
mllib_ImageDivShift2_Inp(3MLIB)	366
mllib_ImageDivShift(3MLIB)	367
mllib_ImageErode4(3MLIB)	368
mllib_ImageErode4_Fp(3MLIB)	369
mllib_ImageErode8(3MLIB)	370
mllib_ImageErode8_Fp(3MLIB)	371
mllib_ImageExp(3MLIB)	372
mllib_ImageExp_Fp(3MLIB)	373
mllib_ImageExp_Fp_Inp(3MLIB)	374
mllib_ImageExp_Inp(3MLIB)	375
mllib_ImageExtrema2(3MLIB)	376
mllib_ImageExtremaLocations(3MLIB)	378
mllib_ImageFilteredSubsample(3MLIB)	381
mllib_ImageFlipAntiDiag(3MLIB)	384
mllib_ImageFlipAntiDiag_Fp(3MLIB)	385
mllib_ImageFlipMainDiag(3MLIB)	386
mllib_ImageFlipMainDiag_Fp(3MLIB)	387
mllib_ImageFlipX(3MLIB)	388
mllib_ImageFlipX_Fp(3MLIB)	389
mllib_ImageFlipY(3MLIB)	390
mllib_ImageFlipY_Fp(3MLIB)	391
mllib_ImageFourierTransform(3MLIB)	392
mllib_ImageGetBitOffset(3MLIB)	394
mllib_ImageGetChannels(3MLIB)	395
mllib_ImageGetData(3MLIB)	396
mllib_ImageGetFlags(3MLIB)	397
mllib_ImageGetFormat(3MLIB)	398
mllib_ImageGetHeight(3MLIB)	399
mllib_ImageGetPaddings(3MLIB)	400
mllib_ImageGetStride(3MLIB)	401
mllib_ImageGetType(3MLIB)	402
mllib_ImageGetWidth(3MLIB)	403
mllib_ImageGradient3x3(3MLIB)	404
mllib_ImageGradient3x3_Fp(3MLIB)	406
mllib_ImageGradientMxN(3MLIB)	408

mllib_ImageGradientMxN_Fp(3MLIB)	410
mllib_ImageGridWarp(3MLIB)	412
mllib_ImageGridWarp_Fp(3MLIB)	415
mllib_ImageGridWarpTable(3MLIB)	418
mllib_ImageGridWarpTable_Fp(3MLIB)	421
mllib_ImageHistogram2(3MLIB)	424
mllib_ImageHistogram(3MLIB)	426
mllib_ImageInterpTableCreate(3MLIB)	427
mllib_ImageInterpTableDelete(3MLIB)	429
mllib_ImageInvert(3MLIB)	430
mllib_ImageInvert_Fp(3MLIB)	431
mllib_ImageInvert_Fp_Inp(3MLIB)	432
mllib_ImageInvert_Inp(3MLIB)	433
mllib_ImageIsNotAligned2(3MLIB)	434
mllib_ImageIsNotAligned4(3MLIB)	435
mllib_ImageIsNotAligned64(3MLIB)	436
mllib_ImageIsNotAligned8(3MLIB)	437
mllib_ImageIsNotHeight2X(3MLIB)	438
mllib_ImageIsNotHeight4X(3MLIB)	439
mllib_ImageIsNotHeight8X(3MLIB)	440
mllib_ImageIsNotOneDvector(3MLIB)	441
mllib_ImageIsNotStride8X(3MLIB)	442
mllib_ImageIsNotWidth2X(3MLIB)	443
mllib_ImageIsNotWidth4X(3MLIB)	444
mllib_ImageIsNotWidth8X(3MLIB)	445
mllib_ImageIsUserAllocated(3MLIB)	446
mllib_ImageLog(3MLIB)	447
mllib_ImageLog_Fp(3MLIB)	448
mllib_ImageLog_Fp_Inp(3MLIB)	449
mllib_ImageLog_Inp(3MLIB)	450
mllib_ImageLookUp2(3MLIB)	451
mllib_ImageLookUp(3MLIB)	453
mllib_ImageLookUp_Inp(3MLIB)	455
mllib_ImageLookUpMask(3MLIB)	456
mllib_ImageMax(3MLIB)	458
mllib_ImageMaxFilter3x3(3MLIB)	459

mllib_ImageMaxFilter3x3_Fp(3MLIB)	461
mllib_ImageMaxFilter5x5(3MLIB)	463
mllib_ImageMaxFilter5x5_Fp(3MLIB)	465
mllib_ImageMaxFilter7x7(3MLIB)	467
mllib_ImageMaxFilter7x7_Fp(3MLIB)	469
mllib_ImageMax_Fp(3MLIB)	471
mllib_ImageMax_Fp_Inp(3MLIB)	472
mllib_ImageMaximum(3MLIB)	473
mllib_ImageMaximum_Fp(3MLIB)	474
mllib_ImageMax_Inp(3MLIB)	475
mllib_ImageMean(3MLIB)	476
mllib_ImageMean_Fp(3MLIB)	477
mllib_ImageMedianFilter3x3(3MLIB)	478
mllib_ImageMedianFilter3x3_Fp(3MLIB)	480
mllib_ImageMedianFilter3x3_US(3MLIB)	482
mllib_ImageMedianFilter5x5(3MLIB)	484
mllib_ImageMedianFilter5x5_Fp(3MLIB)	486
mllib_ImageMedianFilter5x5_US(3MLIB)	488
mllib_ImageMedianFilter7x7(3MLIB)	490
mllib_ImageMedianFilter7x7_Fp(3MLIB)	492
mllib_ImageMedianFilter7x7_US(3MLIB)	494
mllib_ImageMedianFilterMxN(3MLIB)	496
mllib_ImageMedianFilterMxN_Fp(3MLIB)	498
mllib_ImageMedianFilterMxN_US(3MLIB)	500
mllib_ImageMin(3MLIB)	502
mllib_ImageMinFilter3x3(3MLIB)	503
mllib_ImageMinFilter3x3_Fp(3MLIB)	505
mllib_ImageMinFilter5x5(3MLIB)	507
mllib_ImageMinFilter5x5_Fp(3MLIB)	509
mllib_ImageMinFilter7x7(3MLIB)	511
mllib_ImageMinFilter7x7_Fp(3MLIB)	513
mllib_ImageMin_Fp(3MLIB)	515
mllib_ImageMin_Fp_Inp(3MLIB)	516
mllib_ImageMinimum(3MLIB)	517
mllib_ImageMinimum_Fp(3MLIB)	518
mllib_ImageMin_Inp(3MLIB)	519

mllib_ImageMoment2(3MLIB)	520
mllib_ImageMoment2_Fp(3MLIB)	521
mllib_ImageMulAlpha(3MLIB)	522
mllib_ImageMulAlpha_Fp(3MLIB)	524
mllib_ImageMulAlpha_Fp_Inp(3MLIB)	525
mllib_ImageMulAlpha_Inp(3MLIB)	526
mllib_ImageMul_Fp(3MLIB)	527
mllib_ImageMul_Fp_Inp(3MLIB)	528
mllib_ImageMulShift(3MLIB)	529
mllib_ImageMulShift_Inp(3MLIB)	530
mllib_ImageNormCrossCorrel(3MLIB)	531
mllib_ImageNormCrossCorrel_Fp(3MLIB)	534
mllib_ImageNot(3MLIB)	537
mllib_ImageNotAnd(3MLIB)	538
mllib_ImageNotAnd_Inp(3MLIB)	539
mllib_ImageNot_Inp(3MLIB)	540
mllib_ImageNotOr(3MLIB)	541
mllib_ImageNotOr_Inp(3MLIB)	542
mllib_ImageNotXor(3MLIB)	543
mllib_ImageNotXor_Inp(3MLIB)	544
mllib_ImageOr(3MLIB)	545
mllib_ImageOr_Inp(3MLIB)	546
mllib_ImageOrNot1_Inp(3MLIB)	547
mllib_ImageOrNot2_Inp(3MLIB)	548
mllib_ImageOrNot(3MLIB)	549
mllib_ImagePolynomialWarp(3MLIB)	550
mllib_ImagePolynomialWarp_Fp(3MLIB)	553
mllib_ImagePolynomialWarpTable(3MLIB)	556
mllib_ImagePolynomialWarpTable_Fp(3MLIB)	559
mllib_ImageRankFilter3x3(3MLIB)	562
mllib_ImageRankFilter3x3_Fp(3MLIB)	564
mllib_ImageRankFilter3x3_US(3MLIB)	566
mllib_ImageRankFilter5x5(3MLIB)	568
mllib_ImageRankFilter5x5_Fp(3MLIB)	570
mllib_ImageRankFilter5x5_US(3MLIB)	572
mllib_ImageRankFilter7x7(3MLIB)	574

mllib_ImageRankFilter7x7_Fp(3MLIB)	576
mllib_ImageRankFilter7x7_US(3MLIB)	578
mllib_ImageRankFilterMxN(3MLIB)	580
mllib_ImageRankFilterMxN_Fp(3MLIB)	582
mllib_ImageRankFilterMxN_US(3MLIB)	584
mllib_ImageReformat(3MLIB)	586
mllib_ImageReplaceColor(3MLIB)	589
mllib_ImageReplaceColor_Fp(3MLIB)	590
mllib_ImageReplaceColor_Fp_Inp(3MLIB)	591
mllib_ImageReplaceColor_Inp(3MLIB)	592
mllib_ImageResetStruct(3MLIB)	593
mllib_ImageResetSubimageStruct(3MLIB)	595
mllib_ImageRotate180(3MLIB)	597
mllib_ImageRotate180_Fp(3MLIB)	598
mllib_ImageRotate270(3MLIB)	599
mllib_ImageRotate270_Fp(3MLIB)	600
mllib_ImageRotate(3MLIB)	601
mllib_ImageRotate90(3MLIB)	603
mllib_ImageRotate90_Fp(3MLIB)	604
mllib_ImageRotate_Fp(3MLIB)	605
mllib_ImageRotateIndex(3MLIB)	607
mllib_ImageScalarBlend(3MLIB)	609
mllib_ImageScalarBlend_Fp(3MLIB)	610
mllib_ImageScalarBlend_Fp_Inp(3MLIB)	611
mllib_ImageScalarBlend_Inp(3MLIB)	612
mllib_ImageScale2(3MLIB)	613
mllib_ImageScale2_Inp(3MLIB)	615
mllib_ImageScale(3MLIB)	616
mllib_ImageScale_Fp(3MLIB)	618
mllib_ImageScale_Fp_Inp(3MLIB)	620
mllib_ImageScale_Inp(3MLIB)	621
mllib_ImageSConv3x3(3MLIB)	622
mllib_ImageSConv3x3_Fp(3MLIB)	624
mllib_ImageSConv5x5(3MLIB)	626
mllib_ImageSConv5x5_Fp(3MLIB)	628
mllib_ImageSConv7x7(3MLIB)	630

mllib_ImageSConv7x7_Fp(3MLIB)	632
mllib_ImageSConvKernelConvert(3MLIB)	634
mllib_ImageSetFormat(3MLIB)	635
mllib_ImageSetPaddings(3MLIB)	636
mllib_ImageSetStruct(3MLIB)	638
mllib_ImageSetSubimageStruct(3MLIB)	640
mllib_ImageSobel(3MLIB)	642
mllib_ImageSqr_Fp(3MLIB)	644
mllib_ImageSqr_Fp_Inp(3MLIB)	645
mllib_ImageSqrShift(3MLIB)	646
mllib_ImageSqrShift_Inp(3MLIB)	647
mllib_ImageStdDev(3MLIB)	648
mllib_ImageStdDev_Fp(3MLIB)	649
mllib_ImageSub1_Fp_Inp(3MLIB)	650
mllib_ImageSub1_Inp(3MLIB)	651
mllib_ImageSub2_Fp_Inp(3MLIB)	652
mllib_ImageSub2_Inp(3MLIB)	653
mllib_ImageSub(3MLIB)	654
mllib_ImageSub_Fp(3MLIB)	655
mllib_ImageSubsampleAverage(3MLIB)	656
mllib_ImageSubsampleBinaryToGray(3MLIB)	658
mllib_ImageTestFlags(3MLIB)	660
mllib_ImageThresh1(3MLIB)	661
mllib_ImageThresh1_Fp(3MLIB)	663
mllib_ImageThresh1_Fp_Inp(3MLIB)	665
mllib_ImageThresh1_Inp(3MLIB)	666
mllib_ImageThresh2(3MLIB)	667
mllib_ImageThresh2_Fp(3MLIB)	668
mllib_ImageThresh2_Fp_Inp(3MLIB)	669
mllib_ImageThresh2_Inp(3MLIB)	670
mllib_ImageThresh3(3MLIB)	671
mllib_ImageThresh3_Fp(3MLIB)	672
mllib_ImageThresh3_Fp_Inp(3MLIB)	673
mllib_ImageThresh3_Inp(3MLIB)	674
mllib_ImageThresh4(3MLIB)	675
mllib_ImageThresh4_Fp(3MLIB)	677

mllib_ImageThresh4_Fp_Inp(3MLIB)	679
mllib_ImageThresh4_Inp(3MLIB)	681
mllib_ImageThresh5(3MLIB)	683
mllib_ImageThresh5_Fp(3MLIB)	684
mllib_ImageThresh5_Fp_Inp(3MLIB)	685
mllib_ImageThresh5_Inp(3MLIB)	686
mllib_ImageXor(3MLIB)	687
mllib_ImageXor_Inp(3MLIB)	688
mllib_ImageXProj(3MLIB)	689
mllib_ImageXProj_Fp(3MLIB)	690
mllib_ImageYProj(3MLIB)	691
mllib_ImageYProj_Fp(3MLIB)	692
mllib_ImageZoom(3MLIB)	693
mllib_ImageZoomBlend(3MLIB)	695
mllib_ImageZoom_Fp(3MLIB)	697
mllib_ImageZoomIn2X(3MLIB)	699
mllib_ImageZoomIn2X_Fp(3MLIB)	700
mllib_ImageZoomIn2XIndex(3MLIB)	701
mllib_ImageZoomIndex(3MLIB)	703
mllib_ImageZoomOut2X(3MLIB)	705
mllib_ImageZoomOut2X_Fp(3MLIB)	706
mllib_ImageZoomOut2XIndex(3MLIB)	707
mllib_ImageZoomTranslate(3MLIB)	709
mllib_ImageZoomTranslateBlend(3MLIB)	711
mllib_ImageZoomTranslate_Fp(3MLIB)	714
mllib_ImageZoomTranslateTable(3MLIB)	716
mllib_ImageZoomTranslateTableBlend(3MLIB)	718
mllib_ImageZoomTranslateTable_Fp(3MLIB)	721
mllib_ImageZoomTranslateToGray(3MLIB)	723
mllib_malloc(3MLIB)	725
mllib_MatrixAddS_U8_Mod(3MLIB)	726
mllib_MatrixAddS_U8_U8_Mod(3MLIB)	728
mllib_MatrixAdd_U8_Mod(3MLIB)	731
mllib_MatrixAdd_U8_U8_Mod(3MLIB)	733
mllib_MatrixAve_U8(3MLIB)	736
mllib_MatrixAve_U8_U8(3MLIB)	738

mllib_MatrixMaximumMag_U8C(3MLIB)	740
mllib_MatrixMaximum_U8(3MLIB)	741
mllib_MatrixMinimumMag_U8C(3MLIB)	743
mllib_MatrixMinimum_U8(3MLIB)	744
mllib_MatrixMulShift_S16_S16_Mod(3MLIB)	746
 Multimedia Library Functions - Part 4	 749
mllib_MatrixMulSShift_U8_Mod(3MLIB)	750
mllib_MatrixMulSShift_U8_U8_Mod(3MLIB)	753
mllib_MatrixMulS_U8_Mod(3MLIB)	756
mllib_MatrixMulS_U8_U8_Mod(3MLIB)	758
mllib_MatrixMul_U8_U8_Mod(3MLIB)	761
mllib_MatrixScale_U8_Mod(3MLIB)	765
mllib_MatrixScale_U8_U8_Mod(3MLIB)	767
mllib_MatrixSubS_U8_Mod(3MLIB)	771
mllib_MatrixSubS_U8_U8_Mod(3MLIB)	773
mllib_MatrixSub_U8_Mod(3MLIB)	776
mllib_MatrixSub_U8_U8_Mod(3MLIB)	778
mllib_MatrixTranspose_U8(3MLIB)	781
mllib_MatrixTranspose_U8_U8(3MLIB)	783
mllib_MatrixUnit_U8(3MLIB)	785
mllib_memcpy(3MLIB)	787
mllib_memmove(3MLIB)	788
mllib_memset(3MLIB)	789
mllib_realloc(3MLIB)	790
mllib_SignalADPCM2Bits2Linear(3MLIB)	791
mllib_SignalADPCM3Bits2Linear(3MLIB)	792
mllib_SignalADPCM4Bits2Linear(3MLIB)	793
mllib_SignalADPCM5Bits2Linear(3MLIB)	794
mllib_SignalADPCMFree(3MLIB)	795
mllib_SignalADPCMInit(3MLIB)	796
mllib_SignalALaw2Linear(3MLIB)	797
mllib_SignalALaw2uLaw(3MLIB)	798
mllib_SignalAutoCorrel_S16(3MLIB)	799
mllib_SignalCepstral_F32(3MLIB)	801

mllib_SignalCepstralFree_S16(3MLIB)	803
mllib_SignalCepstralInit_S16(3MLIB)	804
mllib_SignalCepstral_S16(3MLIB)	805
mllib_SignalCepstral_S16_Adp(3MLIB)	807
mllib_SignalConvertShift_F32_U8(3MLIB)	809
mllib_SignalConvertShift_U8_S8_Sat(3MLIB)	811
mllib_SignalConv_S16_S16_Sat(3MLIB)	815
mllib_SignalCrossCorrel_S16(3MLIB)	817
mllib_SignalDownSample_S16_S16(3MLIB)	819
mllib_SignalDTWKScalar_F32(3MLIB)	821
mllib_SignalDTWKScalarFree_S16(3MLIB)	825
mllib_SignalDTWKScalarInit_F32(3MLIB)	826
mllib_SignalDTWKScalarInit_S16(3MLIB)	827
mllib_SignalDTWKScalarPath_S16(3MLIB)	828
mllib_SignalDTWKScalar_S16(3MLIB)	832
mllib_SignalDTWKVector_F32(3MLIB)	836
mllib_SignalDTWKVectorFree_S16(3MLIB)	840
mllib_SignalDTWKVectorInit_F32(3MLIB)	841
mllib_SignalDTWKVectorInit_S16(3MLIB)	843
mllib_SignalDTWKVectorPath_S16(3MLIB)	845
mllib_SignalDTWKVector_S16(3MLIB)	849
mllib_SignalDTWScalar_F32(3MLIB)	853
mllib_SignalDTWScalarFree_S16(3MLIB)	857
mllib_SignalDTWScalarInit_F32(3MLIB)	858
mllib_SignalDTWScalarInit_S16(3MLIB)	859
mllib_SignalDTWScalarPath_F32(3MLIB)	860
mllib_SignalDTWScalarPath_S16(3MLIB)	864
mllib_SignalDTWScalar_S16(3MLIB)	868
mllib_SignalDTWVector_F32(3MLIB)	872
mllib_SignalDTWVectorFree_S16(3MLIB)	876
mllib_SignalDTWVectorInit_F32(3MLIB)	877
mllib_SignalDTWVectorInit_S16(3MLIB)	878
mllib_SignalDTWVectorPath_F32(3MLIB)	880
mllib_SignalDTWVectorPath_S16(3MLIB)	884
mllib_SignalDTWVector_S16(3MLIB)	888
mllib_SignalEmphasizeFree_S16_S16(3MLIB)	892

mllib_SignalEmphasizeInit_S16_S16(3MLIB)	893
mllib_SignalEmphasize_S16_S16_Sat(3MLIB)	894
mllib_SignalFFT_1(3MLIB)	896
mllib_SignalFFT_2(3MLIB)	899
mllib_SignalFFT_3(3MLIB)	902
mllib_SignalFFT_4(3MLIB)	905
mllib_SignalFFTW_1(3MLIB)	908
mllib_SignalFFTW_2(3MLIB)	911
mllib_SignalFFTW_3(3MLIB)	914
mllib_SignalFFTW_4(3MLIB)	917
mllib_SignalFIR_F32_F32(3MLIB)	920
mllib_SignalFIR_F32S_F32S(3MLIB)	921
mllib_SignalFIRFree_F32_F32(3MLIB)	922
mllib_SignalFIRFree_F32S_F32S(3MLIB)	923
mllib_SignalFIRFree_S16_S16(3MLIB)	924
mllib_SignalFIRInit_F32_F32(3MLIB)	925
mllib_SignalFIRInit_F32S_F32S(3MLIB)	926
mllib_SignalFIRInit_S16_S16(3MLIB)	927
mllib_SignalFIR_S16_S16_Sat(3MLIB)	928
mllib_SignalGaussNoise_F32(3MLIB)	929
mllib_SignalGaussNoiseFree_F32(3MLIB)	930
mllib_SignalGaussNoiseFree_S16(3MLIB)	931
mllib_SignalGaussNoiseInit_F32(3MLIB)	932
mllib_SignalGaussNoiseInit_S16(3MLIB)	933
mllib_SignalGaussNoise_S16(3MLIB)	934
mllib_SignalGenBartlett_F32(3MLIB)	935
mllib_SignalGenBartlett_S16(3MLIB)	936
mllib_SignalGenBlackman_F32(3MLIB)	937
mllib_SignalGenBlackman_S16(3MLIB)	938
mllib_SignalGenHamming_F32(3MLIB)	939
mllib_SignalGenHamming_S16(3MLIB)	940
mllib_SignalGenHanning_F32(3MLIB)	941
mllib_SignalGenHanning_S16(3MLIB)	942
mllib_SignalGenKaiser_F32(3MLIB)	943
mllib_SignalGenKaiser_S16(3MLIB)	944
mllib_SignalIFFT_1(3MLIB)	945

mllib_SignalIFFT_2(3MLIB)	948
mllib_SignalIFFT_3(3MLIB)	951
mllib_SignalIFFT_4(3MLIB)	954
mllib_SignalIFFTW_1(3MLIB)	957

Multimedia Library Functions - Part 5**961**

mllib_SignalIFFTW_2(3MLIB)	962
mllib_SignalIFFTW_3(3MLIB)	965
mllib_SignalIFFTW_4(3MLIB)	968
mllib_SignalIIR_Biquad_S16_S16_Sat(3MLIB)	971
mllib_SignalIIRFree_Biquad_F32_F32(3MLIB)	973
mllib_SignalIIRFree_Biquad_S16_S16(3MLIB)	974
mllib_SignalIIRFree_P4_F32_F32(3MLIB)	975
mllib_SignalIIRFree_P4_S16_S16(3MLIB)	976
mllib_SignalIIRInit_Biquad_F32_F32(3MLIB)	977
mllib_SignalIIRInit_Biquad_S16_S16(3MLIB)	978
mllib_SignalIIRInit_P4_F32_F32(3MLIB)	979
mllib_SignalIIRInit_P4_S16_S16(3MLIB)	980
mllib_SignalIIR_P4_S16_S16_Sat(3MLIB)	981
mllib_SignalIMDCT_D64(3MLIB)	984
mllib_SignalIMDCT_F32(3MLIB)	985
mllib_SignalIMDCTSplit_D64(3MLIB)	986
mllib_SignalIMDCTSplit_F32(3MLIB)	987
mllib_SignalLimit(3MLIB)	988
mllib_SignalLinear2ADPCM2Bits(3MLIB)	990
mllib_SignalLinear2ADPCM3Bits(3MLIB)	991
mllib_SignalLinear2ADPCM4Bits(3MLIB)	992
mllib_SignalLinear2ADPCM5Bits(3MLIB)	993
mllib_SignalLinear2ALaw(3MLIB)	994
mllib_SignalLinear2uLaw(3MLIB)	995
mllib_SignalLMSFilter(3MLIB)	996
mllib_SignalLPC2Cepstral_F32(3MLIB)	999
mllib_SignalLPC2Cepstral_S16(3MLIB)	1001
mllib_SignalLPC2Cepstral_S16_Adp(3MLIB)	1003
mllib_SignalLPC2LSP_F32(3MLIB)	1005

mllib_SignalLPC2LSP_S16(3MLIB)	1007
mllib_SignalLPCAutoCorrel_F32(3MLIB)	1009
mllib_SignalLPCAutoCorrelFree_S16(3MLIB)	1011
mllib_SignalLPCAutoCorrelGetEnergy_F32(3MLIB)	1012
mllib_SignalLPCAutoCorrelGetEnergy_S16(3MLIB)	1014
mllib_SignalLPCAutoCorrelGetPARCOR_F32(3MLIB)	1016
mllib_SignalLPCAutoCorrelGetPARCOR_S16(3MLIB)	1018
mllib_SignalLPCAutoCorrelInit_S16(3MLIB)	1020
mllib_SignalLPCAutoCorrel_S16(3MLIB)	1021
mllib_SignalLPCCovariance_F32(3MLIB)	1023
mllib_SignalLPCCovarianceFree_S16(3MLIB)	1025
mllib_SignalLPCCovarianceInit_S16(3MLIB)	1026
mllib_SignalLPCCovariance_S16(3MLIB)	1027
mllib_SignalLPCPerceptWeight_F32(3MLIB)	1029
mllib_SignalLPCPerceptWeightFree_S16(3MLIB)	1031
mllib_SignalLPCPerceptWeightInit_S16(3MLIB)	1032
mllib_SignalLPCPerceptWeight_S16(3MLIB)	1033
mllib_SignalLPCPitchAnalyze_F32(3MLIB)	1035
mllib_SignalLPCPitchAnalyze_S16(3MLIB)	1037
mllib_SignalLSP2LPC_F32(3MLIB)	1039
mllib_SignalLSP2LPC_S16(3MLIB)	1041
mllib_SignalMelCepstral_F32(3MLIB)	1043
mllib_SignalMelCepstralFree_S16(3MLIB)	1045
mllib_SignalMelCepstralInit_S16(3MLIB)	1046
mllib_SignalMelCepstral_S16(3MLIB)	1048
mllib_SignalMelCepstral_S16_AdP(3MLIB)	1050
mllib_SignalMerge_F32S_F32(3MLIB)	1052
mllib_SignalMerge_S16S_S16(3MLIB)	1053
mllib_SignalMulBartlett_F32(3MLIB)	1054
mllib_SignalMulBartlett_F32_F32(3MLIB)	1055
mllib_SignalMulBartlett_S16(3MLIB)	1056
mllib_SignalMulBartlett_S16_S16(3MLIB)	1057
mllib_SignalMulBlackman_F32(3MLIB)	1058
mllib_SignalMulBlackman_F32_F32(3MLIB)	1059
mllib_SignalMulBlackman_S16(3MLIB)	1060
mllib_SignalMulBlackman_S16_S16(3MLIB)	1061

mllib_SignalMul_F32(3MLIB)	1062
mllib_SignalMul_F32_F32(3MLIB)	1063
mllib_SignalMulHamming_F32(3MLIB)	1064
mllib_SignalMulHamming_F32_F32(3MLIB)	1065
mllib_SignalMulHamming_S16(3MLIB)	1066
mllib_SignalMulHamming_S16_S16(3MLIB)	1067
mllib_SignalMulHanning_F32(3MLIB)	1068
mllib_SignalMulHanning_F32_F32(3MLIB)	1069
mllib_SignalMulHanning_S16(3MLIB)	1070
mllib_SignalMulHanning_S16_S16(3MLIB)	1071
mllib_SignalMulKaiser_F32(3MLIB)	1072
mllib_SignalMulKaiser_F32_F32(3MLIB)	1073
mllib_SignalMulKaiser_S16(3MLIB)	1074
mllib_SignalMulKaiser_S16_S16(3MLIB)	1075
mllib_SignalMulRectangular_F32(3MLIB)	1076
mllib_SignalMulRectangular_F32_F32(3MLIB)	1077
mllib_SignalMulRectangular_S16(3MLIB)	1078
mllib_SignalMulRectangular_S16_S16(3MLIB)	1079
mllib_SignalMul_S16_S16_Sat(3MLIB)	1080
mllib_SignalMul_S16_Sat(3MLIB)	1081
mllib_SignalMulSAdd_F32(3MLIB)	1082
mllib_SignalMulSAdd_F32_F32(3MLIB)	1083
mllib_SignalMulSAdd_S16_S16_Sat(3MLIB)	1084
mllib_SignalMulSAdd_S16_Sat(3MLIB)	1085
mllib_SignalMulS_F32(3MLIB)	1086
mllib_SignalMulS_F32_F32(3MLIB)	1087
mllib_SignalMulShift_S16_S16_Sat(3MLIB)	1088
mllib_SignalMulShift_S16_Sat(3MLIB)	1089
mllib_SignalMulS_S16_S16_Sat(3MLIB)	1090
mllib_SignalMulS_S16_Sat(3MLIB)	1091
mllib_SignalMulSShiftAdd_S16_S16_Sat(3MLIB)	1092
mllib_SignalMulSShiftAdd_S16_Sat(3MLIB)	1093
mllib_SignalMulSShift_S16_S16_Sat(3MLIB)	1094
mllib_SignalMulSShift_S16_Sat(3MLIB)	1095
mllib_SignalMulWindow_F32(3MLIB)	1096
mllib_SignalMulWindow_F32_F32(3MLIB)	1097

mllib_SignalMulWindow_F32S(3MLIB)	1098
mllib_SignalMulWindow_F32S_F32S(3MLIB)	1099
mllib_SignalMulWindow_S16(3MLIB)	1100
mllib_SignalMulWindow_S16_S16(3MLIB)	1101
mllib_SignalNLMSFilter(3MLIB)	1102
mllib_SignalQuant2_S16_F32(3MLIB)	1105
mllib_SignalQuant2_S16S_F32S(3MLIB)	1106
mllib_SignalQuant_S16_F32(3MLIB)	1107
mllib_SignalQuant_S16S_F32S(3MLIB)	1108
mllib_SignalQuant_U8_F32(3MLIB)	1109
mllib_SignalQuant_U8_S16(3MLIB)	1110
mllib_SignalQuant_U8S_F32S(3MLIB)	1111
mllib_SignalReSampleFIR_F32_F32(3MLIB)	1112
mllib_SignalReSampleFIR_F32S_F32S(3MLIB)	1113
mllib_SignalReSampleFIRFree_F32_F32(3MLIB)	1114
mllib_SignalReSampleFIRFree_F32S_F32S(3MLIB)	1115
mllib_SignalReSampleFIRFree_S16_S16(3MLIB)	1116
mllib_SignalReSampleFIRInit_S16_S16(3MLIB)	1117
mllib_SignalReSampleFIR_S16_S16_Sat(3MLIB)	1119
mllib_SignalSineWave_F32(3MLIB)	1120
mllib_SignalSineWaveFree_F32(3MLIB)	1121
mllib_SignalSineWaveFree_S16(3MLIB)	1122
mllib_SignalSineWaveInit_F32(3MLIB)	1123
mllib_SignalSineWaveInit_S16(3MLIB)	1124
mllib_SignalSineWave_S16(3MLIB)	1125
mllib_SignalSplit_F32_F32S(3MLIB)	1126
mllib_SignalSplit_S16_S16S(3MLIB)	1127
mllib_SignalLuLaw2ALaw(3MLIB)	1128
mllib_SignalLuLaw2Linear(3MLIB)	1129
mllib_SignalUpSampleFIR_F32_F32(3MLIB)	1130
mllib_SignalUpSampleFIR_F32S_F32S(3MLIB)	1131
mllib_SignalUpSampleFIRFree_F32_F32(3MLIB)	1132
mllib_SignalUpSampleFIRFree_F32S_F32S(3MLIB)	1133
mllib_SignalUpSampleFIRFree_S16_S16(3MLIB)	1134
mllib_SignalUpSampleFIRInit_F32_F32(3MLIB)	1135
mllib_SignalUpSampleFIRInit_F32S_F32S(3MLIB)	1136

mllib_SignalUpSampleFIRInit_S16_S16(3MLIB)	1137
mllib_SignalUpSampleFIR_S16_S16_Sat(3MLIB)	1138
mllib_SignalUpSample_S16_S16(3MLIB)	1139
mllib_SignalWhiteNoise_F32(3MLIB)	1141
mllib_SignalWhiteNoiseFree_F32(3MLIB)	1142
mllib_SignalWhiteNoiseFree_S16(3MLIB)	1143
mllib_SignalWhiteNoiseInit_F32(3MLIB)	1144
mllib_SignalWhiteNoiseInit_S16(3MLIB)	1145
mllib_SignalWhiteNoise_S16(3MLIB)	1146
mllib_VectorAddS_U8_Mod(3MLIB)	1147
mllib_VectorAddS_U8_U8_Mod(3MLIB)	1149
mllib_VectorAdd_U8_Mod(3MLIB)	1152
mllib_VectorAdd_U8_U8_Mod(3MLIB)	1154
mllib_VectorAng_U8C(3MLIB)	1157
mllib_VectorAve_U8(3MLIB)	1158
mllib_VectorAve_U8_U8(3MLIB)	1160
mllib_VectorConjRev_S8C_S8C_Sat(3MLIB)	1162
mllib_VectorConj_S8C_S8C_Sat(3MLIB)	1163
mllib_VectorConj_S8C_Sat(3MLIB)	1164
mllib_VectorConjSymExt_S8C_S8C_Sat(3MLIB)	1165
 Multimedia Library Functions - Part 6	 1167
mllib_VectorConvert_U8_S8_Mod(3MLIB)	1168
mllib_VectorCopy_U8(3MLIB)	1173
mllib_VectorDistance_U8_Sat(3MLIB)	1175
mllib_VectorDotProd_U8_Sat(3MLIB)	1176
mllib_VectorMag_U8C(3MLIB)	1178
mllib_VectorMaximumMag_U8C(3MLIB)	1179
mllib_VectorMaximum_U8(3MLIB)	1180
mllib_VectorMerge_U8C_U8(3MLIB)	1181
mllib_VectorMinimumMag_U8C(3MLIB)	1183
mllib_VectorMinimum_U8(3MLIB)	1184
mllib_VectorMulMShift_S16_S16_Mod(3MLIB)	1185
mllib_VectorMulM_U8_U8_Mod(3MLIB)	1187
mllib_VectorMulSAdd_U8_Mod(3MLIB)	1190

mllib_VectorMulSAdd_U8_U8_Mod(3MLIB)	1192
mllib_VectorMulShift_U8_Mod(3MLIB)	1196
mllib_VectorMulShift_U8_U8_Mod(3MLIB)	1198
mllib_VectorMulSShift_U8_Mod(3MLIB)	1200
mllib_VectorMulSShift_U8_U8_Mod(3MLIB)	1202
mllib_VectorMulS_U8_Mod(3MLIB)	1204
mllib_VectorMulS_U8_U8_Mod(3MLIB)	1206
mllib_VectorMul_U8_Mod(3MLIB)	1209
mllib_VectorMul_U8_U8_Mod(3MLIB)	1211
mllib_VectorNorm_U8_Sat(3MLIB)	1214
mllib_VectorReverseByteOrder(3MLIB)	1215
mllib_VectorReverseByteOrder_Inp(3MLIB)	1216
mllib_VectorReverseByteOrder_S16(3MLIB)	1217
mllib_VectorReverseByteOrder_S16_S16(3MLIB)	1219
mllib_VectorScale_U8_Mod(3MLIB)	1221
mllib_VectorScale_U8_U8_Mod(3MLIB)	1223
mllib_VectorSet_U8(3MLIB)	1226
mllib_VectorSplit_U8_U8C(3MLIB)	1228
mllib_VectorSubS_U8_Mod(3MLIB)	1230
mllib_VectorSubS_U8_U8_Mod(3MLIB)	1232
mllib_VectorSub_U8_Mod(3MLIB)	1235
mllib_VectorSub_U8_U8_Mod(3MLIB)	1237
mllib_VectorSumAbsDiff_U8_Sat(3MLIB)	1240
mllib_VectorSumAbs_U8_Sat(3MLIB)	1241
mllib_VectorZero_U8(3MLIB)	1242
mllib_version(3MLIB)	1243
mllib_VideoAddBlock_U8_S16(3MLIB)	1244
mllib_VideoColorABGR2JFIFYCC420(3MLIB)	1245
mllib_VideoColorABGR2JFIFYCC422(3MLIB)	1246
mllib_VideoColorABGR2JFIFYCC444(3MLIB)	1247
 Multimedia Library Functions - Part 7	 1249
mllib_VideoColorABGR2RGB(3MLIB)	1250
mllib_VideoColorABGRint_to_ARGBint(3MLIB)	1251
mllib_VideoColorARGB2JFIFYCC420(3MLIB)	1252

mllib_VideoColorARGB2JFIFYCC422(3MLIB)	1253
mllib_VideoColorARGB2JFIFYCC444(3MLIB)	1254
mllib_VideoColorARGB2RGB(3MLIB)	1255
mllib_VideoColorBGR2JFIFYCC420(3MLIB)	1256
mllib_VideoColorBGR2JFIFYCC422(3MLIB)	1257
mllib_VideoColorBGR2JFIFYCC444(3MLIB)	1258
mllib_VideoColorBGR2JFIFYCC444_S16(3MLIB)	1259
mllib_VideoColorBGRAint_to_ABGRint(3MLIB)	1260
mllib_VideoColorBGRint_to_ABGRint(3MLIB)	1261
mllib_VideoColorBlendABGR(3MLIB)	1263
mllib_VideoColorBlendABGR_Inp(3MLIB)	1265
mllib_VideoColorCMYK2JFIFYCCK444(3MLIB)	1267
mllib_VideoColorJFIFYCC2ABGR444(3MLIB)	1268
mllib_VideoColorJFIFYCC2ARGB444(3MLIB)	1269
mllib_VideoColorJFIFYCC2RGB420(3MLIB)	1270
mllib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB)	1272
mllib_VideoColorJFIFYCC2RGB422(3MLIB)	1273
mllib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB)	1274
mllib_VideoColorJFIFYCC2RGB444(3MLIB)	1275
mllib_VideoColorJFIFYCC2RGB444_S16(3MLIB)	1276
mllib_VideoColorJFIFYCCK2CMYK444(3MLIB)	1277
mllib_VideoColorMerge2(3MLIB)	1278
mllib_VideoColorMerge2_S16(3MLIB)	1279
mllib_VideoColorMerge3(3MLIB)	1280
mllib_VideoColorMerge3_S16(3MLIB)	1281
mllib_VideoColorMerge4(3MLIB)	1282
mllib_VideoColorMerge4_S16(3MLIB)	1283
mllib_VideoColorResizeABGR(3MLIB)	1284
mllib_VideoColorRGB2ABGR(3MLIB)	1286
mllib_VideoColorRGB2ARGB(3MLIB)	1287
mllib_VideoColorRGB2JFIFYCC420(3MLIB)	1288
mllib_VideoColorRGB2JFIFYCC422(3MLIB)	1289
mllib_VideoColorRGB2JFIFYCC444(3MLIB)	1290
mllib_VideoColorRGB2JFIFYCC444_S16(3MLIB)	1291
mllib_VideoColorRGBAint_to_ABGRint(3MLIB)	1292
mllib_VideoColorRGBint_to_ABGRint(3MLIB)	1293

mllib_VideoColorRGBint_to_BGRAint(3MLIB)	1295
mllib_VideoColorRGBseq_to_ABGRint(3MLIB)	1297
mllib_VideoColorRGBXint_to_ABGRint(3MLIB)	1299
mllib_VideoColorRGBXint_to_ARGBint(3MLIB)	1301
mllib_VideoColorSplit2(3MLIB)	1302
mllib_VideoColorSplit2_S16(3MLIB)	1303
mllib_VideoColorSplit3(3MLIB)	1304
mllib_VideoColorSplit3_S16(3MLIB)	1305
mllib_VideoColorSplit4(3MLIB)	1306
mllib_VideoColorSplit4_S16(3MLIB)	1307
mllib_VideoColorUYV444int_to_ABGRint(3MLIB)	1308
mllib_VideoColorUYV444int_to_ARGBint(3MLIB)	1310
mllib_VideoColorUYV444int_to_UYVY422int(3MLIB)	1312
mllib_VideoColorUYV444int_to_YUYV422int(3MLIB)	1314
mllib_VideoColorUYVY422int_to_ABGRint(3MLIB)	1316
mllib_VideoColorUYVY422int_to_ARGBint(3MLIB)	1318
mllib_VideoColorXRGBint_to_ABGRint(3MLIB)	1320
mllib_VideoColorXRGBint_to_ARGBint(3MLIB)	1321
mllib_VideoColorYUV2ABGR411(3MLIB)	1322
mllib_VideoColorYUV2ABGR420(3MLIB)	1324
mllib_VideoColorYUV2ABGR420_w(3MLIB)	1326
mllib_VideoColorYUV2ABGR420_WX2(3MLIB)	1328
mllib_VideoColorYUV2ABGR420_WX3(3MLIB)	1330
mllib_VideoColorYUV2ABGR420_X2(3MLIB)	1332
mllib_VideoColorYUV2ABGR420_X3(3MLIB)	1334
mllib_VideoColorYUV2ABGR422(3MLIB)	1336
mllib_VideoColorYUV2ABGR444(3MLIB)	1338
mllib_VideoColorYUV2ARGB411(3MLIB)	1339
mllib_VideoColorYUV2ARGB420(3MLIB)	1341
mllib_VideoColorYUV2ARGB422(3MLIB)	1343
mllib_VideoColorYUV2ARGB444(3MLIB)	1345
mllib_VideoColorYUV2RGB411(3MLIB)	1346
mllib_VideoColorYUV2RGB420(3MLIB)	1348
mllib_VideoColorYUV2RGB422(3MLIB)	1350
mllib_VideoColorYUV2RGB444(3MLIB)	1352
mllib_VideoColorYUV411seq_to_ABGRint(3MLIB)	1353

mllib_VideoColorYUV411seq_to_ARGBint(3MLIB)	1355
mllib_VideoColorYUV411seq_to_UYVY422int(3MLIB)	1357
mllib_VideoColorYUV411seq_to_YUYV422int(3MLIB)	1359
mllib_VideoColorYUV420seq_to_ABGRint(3MLIB)	1361
mllib_VideoColorYUV420seq_to_ARGBint(3MLIB)	1363
mllib_VideoColorYUV420seq_to_UYVY422int(3MLIB)	1365
mllib_VideoColorYUV420seq_to_YUYV422int(3MLIB)	1367
mllib_VideoColorYUV422seq_to_ABGRint(3MLIB)	1369
mllib_VideoColorYUV422seq_to_ARGBint(3MLIB)	1371
mllib_VideoColorYUV422seq_to_UYVY422int(3MLIB)	1373
mllib_VideoColorYUV422seq_to_YUYV422int(3MLIB)	1375
mllib_VideoColorYUV444int_to_ABGRint(3MLIB)	1377
mllib_VideoColorYUV444int_to_ARGBint(3MLIB)	1379
mllib_VideoColorYUV444int_to_UYVY422int(3MLIB)	1381
mllib_VideoColorYUV444int_to_YUYV422int(3MLIB)	1383
mllib_VideoColorYUV444seq_to_ABGRint(3MLIB)	1385
mllib_VideoColorYUV444seq_to_ARGBint(3MLIB)	1387
mllib_VideoColorYUV444seq_to_UYVY422int(3MLIB)	1389
mllib_VideoColorYUV444seq_to_YUYV422int(3MLIB)	1391
mllib_VideoColorYUYV422int_to_ABGRint(3MLIB)	1393
mllib_VideoColorYUYV422int_to_ARGBint(3MLIB)	1395
mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)	1397
mllib_VideoCopyRefAve_U8_U8(3MLIB)	1399
mllib_VideoCopyRef_S16_U8_16x16(3MLIB)	1401
mllib_VideoCopyRef_S16_U8(3MLIB)	1403
mllib_VideoCopyRef_U8_U8_16x16(3MLIB)	1405
mllib_VideoCopyRef_U8_U8(3MLIB)	1407
mllib_VideoDCT16x16_S16_S16(3MLIB)	1409
mllib_VideoDCT16x16_S16_S16_B10(3MLIB)	1410
mllib_VideoDCT2x2_S16_S16(3MLIB)	1411
mllib_VideoDCT4x4_S16_S16(3MLIB)	1412
mllib_VideoDCT8x8Quantize_S16_S16_B12(3MLIB)	1413
mllib_VideoDCT8x8Quantize_S16_S16_B12_NA(3MLIB)	1414
mllib_VideoDCT8x8Quantize_S16_U8(3MLIB)	1415
mllib_VideoDCT8x8Quantize_S16_U8_NA(3MLIB)	1416
mllib_VideoDCT8x8_S16_S16_B10(3MLIB)	1417

mllib_VideoDCT8x8_S16_S16_B10_NA(3MLIB)	1418
mllib_VideoDCT8x8_S16_S16_B12(3MLIB)	1419
mllib_VideoDCT8x8_S16_U8(3MLIB)	1420
mllib_VideoDCT8x8_S16_U8_NA(3MLIB)	1421
mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12(3MLIB)	1422
mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA(3MLIB)	1423
mllib_VideoDeQuantizeIDCT8x8_U8_S16(3MLIB)	1424
mllib_VideoDeQuantizeIDCT8x8_U8_S16_NA(3MLIB)	1425
mllib_VideoDeQuantizeInit_S16(3MLIB)	1426
mllib_VideoDeQuantize_S16(3MLIB)	1427
mllib_VideoDownSample420(3MLIB)	1428
mllib_VideoDownSample420_S16(3MLIB)	1429
mllib_VideoDownSample422(3MLIB)	1430
mllib_VideoDownSample422_S16(3MLIB)	1431
mllib_VideoH2630verlappedMC_S16_U8(3MLIB)	1432
mllib_VideoH2630verlappedMC_U8_U8(3MLIB)	1435
mllib_VideoIDCT8x8_S16_S16_B12(3MLIB)	1438
mllib_VideoIDCT8x8_S16_S16_B12_NA(3MLIB)	1440
mllib_VideoIDCT8x8_S16_S16_DC(3MLIB)	1442
mllib_VideoIDCT8x8_S16_S16_Q1(3MLIB)	1443
mllib_VideoIDCT8x8_S16_S16_Q1_Mismatch(3MLIB)	1444
mllib_VideoIDCT8x8_U8_S16(3MLIB)	1445
mllib_VideoIDCT8x8_U8_S16_DC(3MLIB)	1446
mllib_VideoIDCT8x8_U8_S16_NA(3MLIB)	1447
mllib_VideoIDCT8x8_U8_S16_Q1(3MLIB)	1448
mllib_VideoIDCT_IEEE_S16_S16(3MLIB)	1449
mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)	1450
mllib_VideoInterpAveX_U8_U8(3MLIB)	1452
mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)	1454
mllib_VideoInterpAveXY_U8_U8(3MLIB)	1456
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB)	1458
mllib_VideoInterpAveY_U8_U8(3MLIB)	1460
mllib_VideoInterpX_S16_U8_16x16(3MLIB)	1462
mllib_VideoInterpX_S16_U8(3MLIB)	1464
mllib_VideoInterpX_U8_U8_16x16(3MLIB)	1466
mllib_VideoInterpX_U8_U8(3MLIB)	1468

mllib_VideoInterpXY_S16_U8_16x16(3MLIB)	1470
mllib_VideoInterpXY_S16_U8(3MLIB)	1472
mllib_VideoInterpXY_U8_U8_16x16(3MLIB)	1474
mllib_VideoInterpXY_U8_U8(3MLIB)	1476
mllib_VideoInterpX_Y_XY_U8_U8(3MLIB)	1478
mllib_VideoInterpY_S16_U8_16x16(3MLIB)	1479
mllib_VideoInterpY_S16_U8(3MLIB)	1481
mllib_VideoInterpY_U8_U8_16x16(3MLIB)	1483
mllib_VideoInterpY_U8_U8(3MLIB)	1485
mllib_VideoP64Decimate_U8_U8(3MLIB)	1487
mllib_VideoP64Loop_S16_U8(3MLIB)	1489
mllib_VideoP64Loop_U8_U8(3MLIB)	1491
mllib_VideoQuantizeInit_S16(3MLIB)	1493
mllib_VideoQuantize_S16(3MLIB)	1494
mllib_VideoReversibleColorRGB2YUV_U8_U8(3MLIB)	1495
mllib_VideoReversibleColorYUV2RGB_U8_U8(3MLIB)	1497
mllib_VideoSignMagnitudeConvert_S16(3MLIB)	1499
mllib_VideoSignMagnitudeConvert_S16_S16(3MLIB)	1500
mllib_VideoSignMagnitudeConvert_S32(3MLIB)	1501
mllib_VideoSignMagnitudeConvert_S32_S32(3MLIB)	1502
mllib_VideoSumAbsDiff(3MLIB)	1503
mllib_VideoUpSample420(3MLIB)	1504
mllib_VideoUpSample420_Nearest(3MLIB)	1505
mllib_VideoUpSample420_Nearest_S16(3MLIB)	1506
mllib_VideoUpSample420_S16(3MLIB)	1507
mllib_VideoUpSample422(3MLIB)	1508
mllib_VideoUpSample422_Nearest(3MLIB)	1509
mllib_VideoUpSample422_Nearest_S16(3MLIB)	1510
mllib_VideoUpSample422_S16(3MLIB)	1511
mllib_VideoWaveletForwardTwoTenTrans(3MLIB)	1512
mllib_VideoWaveletInverseTwoTenTrans(3MLIB)	1513
mllib_VolumeFindMaxBMask_U8(3MLIB)	1514
mllib_VolumeFindMaxCMask_U8(3MLIB)	1515
mllib_VolumeFindMax_U8(3MLIB)	1516
mllib_VolumeRayCast_Blocked(3MLIB)	1517
mllib_VolumeRayCast_General(3MLIB)	1519

mLib_VolumeWindowLevel(3MLIB) 1522

Preface

Both novice users and those familiar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and [man\(1\)](#) for more information about man pages in general.

NAME	This section gives the names of the commands or functions documented, followed by a brief description of what they do.								
SYNOPSIS	<p>This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.</p> <p>The following special characters are used in this section:</p> <table><tr><td>[]</td><td>Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.</td></tr><tr><td>. . .</td><td>Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename...".</td></tr><tr><td> </td><td>Separator. Only one of the arguments separated by this character can be specified at a time.</td></tr><tr><td>{ }</td><td>Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.</td></tr></table>	[]	Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.	. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename...".		Separator. Only one of the arguments separated by this character can be specified at a time.	{ }	Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
[]	Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.								
. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename...".								
	Separator. Only one of the arguments separated by this character can be specified at a time.								
{ }	Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.								
PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.								
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.								
IOCTL	This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called <code>ioctl</code> and generates its own								

	<p>heading. <code>ioctl</code> calls for a specific device are listed alphabetically (on the man page for that specific device). <code>ioctl</code> calls are used for a particular class of devices all of which have an <code>io</code> ending, such as <code>mtio(7I)</code>.</p>
OPTIONS	<p>This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.</p>
OPERANDS	<p>This section lists the command operands and describes how they affect the actions of the command.</p>
OUTPUT	<p>This section describes the output – standard output, standard error, or output files – generated by the command.</p>
RETURN VALUES	<p>If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.</p>
ERRORS	<p>On failure, most functions place an error code in the global variable <code>errno</code> indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.</p>
USAGE	<p>This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:</p> <ul style="list-style-type: none">CommandsModifiersVariablesExpressionsInput Grammar

EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as <code>example%</code> , or if the user must be superuser, <code>example#</code> . Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See attributes(5) for more information.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

Multimedia Library Functions - Part 1

Name mllib_free – free a block of bytes

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

void mllib_free(void *ptr);
```

Description The `mllib_free()` function frees a block of bytes previously allocated by `mllib_malloc()` or `mllib_realloc()`.

This function is a wrapper of the standard C function `free()`.

Parameters The function takes the following arguments:

ptr Pointer to a previously allocated block.

Return Values None.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_malloc\(3MLIB\)](#), [mllib_realloc\(3MLIB\)](#), [malloc\(3C\)](#), [attributes\(5\)](#)

Name mllib_GraphicsBoundaryFill_8, mllib_GraphicsBoundaryFill_32 – boundary fill

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_GraphicsBoundaryFill_8(mllib_image *buffer, mllib_s16 x,
    mllib_s16 y, mllib_s32 c, mllib_s32 c2);

mllib_status mllib_GraphicsBoundaryFill_32(mllib_image *buffer, mllib_s16 x,
    mllib_s16 y, mllib_s32 c, mllib_s32 c2);
```

Description Each of these functions performs boundary fill.

Parameters Each of the functions takes the following arguments:

buffer Pointer to the image into which the function is drawing.

x X coordinate of the starting point.

y Y coordinate of the starting point.

c Color used in the drawing.

c2 Color that defines the filling boundary.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [attributes\(5\)](#)

Name mllib_GraphicsDrawArc, mllib_GraphicsDrawArc_8, mllib_GraphicsDrawArc_32, mllib_GraphicsDrawArc_X_8, mllib_GraphicsDrawArc_X_32, mllib_GraphicsDrawArc_A_8, mllib_GraphicsDrawArc_A_32, mllib_GraphicsDrawArc_B_8, mllib_GraphicsDrawArc_B_32, mllib_GraphicsDrawArc_AB_8, mllib_GraphicsDrawArc_AB_32 – draw arc

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_GraphicsDrawArc_8(mllib_image *buffer, mllib_s16 x,
                                     mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);

mllib_status mllib_GraphicsDrawArc_32(mllib_image *buffer, mllib_s16 x,
                                     mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);

mllib_status mllib_GraphicsDrawArc_X_8(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 c2);

mllib_status mllib_GraphicsDrawArc_X_32(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 c2);

mllib_status mllib_GraphicsDrawArc_A_8(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);

mllib_status mllib_GraphicsDrawArc_A_32(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);

mllib_status mllib_GraphicsDrawArc_B_8(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 a);

mllib_status mllib_GraphicsDrawArc_B_32(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 a);

mllib_status mllib_GraphicsDrawArc_AB_8(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 a);

mllib_status mllib_GraphicsDrawArc_AB_32(mllib_image *buffer, mllib_s16 x,
                                       mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,
                                       mllib_s32 a);
```

Description Each of the `mllib_GraphicsDrawArc_*`() functions draws an arc with the center at (x, y) , radius r , start angle $t1$, and end angle $t2$.

Each of the `mllib_GraphicsDrawArc_X_*`() functions draws an arc in Xor mode as follows:

$$\text{data}[x, y] \hat{=} c \hat{\wedge} c2$$

Each of the `mllib_GraphicsDrawArc_A_*`() functions draws an arc with antialiasing.

Each of the `mllib_GraphicsDrawArc_B_*`() functions draws an arc with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawArc_AB_*`() functions draws an arc with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>x</i>	X coordinate of the center.
<i>y</i>	Y coordinate of the center.
<i>r</i>	Radius of the arc.
<i>t1</i>	Start angle of the arc in radians.
<i>t2</i>	End angle of the arc in radians.
<i>c</i>	Color used in the drawing.
<i>c2</i>	Alternation color.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawCircle\(3MLIB\)](#), [mllib_GraphicsDrawEllipse\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawCircle, mllib_GraphicsDrawCircle_8, mllib_GraphicsDrawCircle_32, mllib_GraphicsDrawCircle_X_8, mllib_GraphicsDrawCircle_X_32, mllib_GraphicsDrawCircle_A_8, mllib_GraphicsDrawCircle_A_32, mllib_GraphicsDrawCircle_B_8, mllib_GraphicsDrawCircle_B_32, mllib_GraphicsDrawCircle_AB_8, mllib_GraphicsDrawCircle_AB_32 – draw circle

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawCircle_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawCircle_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawCircle_X_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawCircle_X_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawCircle_A_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawCircle_A_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawCircle_B_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawCircle_B_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawCircle_AB_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawCircle_AB_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
```

Description Each of the `mllib_GraphicsDrawCircle_*`() functions draws a circle with the center at (x, y) and radius *r*.

Each of the `mllib_GraphicsDrawCircle_X_*`() functions draws a circle in Xor mode as follows:

$$\text{data}[x,y] \hat{=} c \wedge c2$$

Each of the `mllib_GraphicsDrawCircle_A_*`() functions draws a circle with antialiasing.

Each of the `mllib_GraphicsDrawCircle_B_*`() functions draws a circle with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawCircle_AB_*`() functions draws a circle with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- x* X coordinate of the center.
- y* Y coordinate of the center.
- r* Radius of the arc.
- c* Color used in the drawing.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawArc\(3MLIB\)](#), [mllib_GraphicsDrawEllipse\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawEllipse, mllib_GraphicsDrawEllipse_8, mllib_GraphicsDrawEllipse_32, mllib_GraphicsDrawEllipse_X_8, mllib_GraphicsDrawEllipse_X_32, mllib_GraphicsDrawEllipse_A_8, mllib_GraphicsDrawEllipse_A_32, mllib_GraphicsDrawEllipse_B_8, mllib_GraphicsDrawEllipse_B_32, mllib_GraphicsDrawEllipse_AB_8, mllib_GraphicsDrawEllipse_AB_32 – draw ellipse

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawEllipse_8(mllib_image *buffer, mllib_s16 x,
                                          mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);

mllib_status mllib_GraphicsDrawEllipse_32(mllib_image *buffer, mllib_s16 x,
                                           mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);

mllib_status mllib_GraphicsDrawEllipse_X_8(mllib_image *buffer, mllib_s16 x,
                                           mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                           mllib_s32 c2);

mllib_status mllib_GraphicsDrawEllipse_X_32(mllib_image *buffer, mllib_s16 x,
                                             mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                             mllib_s32 c2);

mllib_status mllib_GraphicsDrawEllipse_A_8(mllib_image *buffer, mllib_s16 x,
                                           mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);

mllib_status mllib_GraphicsDrawEllipse_A_32(mllib_image *buffer, mllib_s16 x,
                                             mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);

mllib_status mllib_GraphicsDrawEllipse_B_8(mllib_image *buffer, mllib_s16 x,
                                           mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                           mllib_s32 alpha);

mllib_status mllib_GraphicsDrawEllipse_B_32(mllib_image *buffer, mllib_s16 x,
                                             mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                             mllib_s32 alpha);

mllib_status mllib_GraphicsDrawEllipse_AB_8(mllib_image *buffer, mllib_s16 x,
                                             mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                             mllib_s32 alpha);

mllib_status mllib_GraphicsDrawEllipse_AB_32(mllib_image *buffer, mllib_s16 x,
                                              mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,
                                              mllib_s32 alpha);
```

Description Each of the `mllib_GraphicsDrawEllipse_*`() functions draws an ellipse with the center at (*x*, *y*), major semiaxis *a*, and minor semiaxis *b*. The angle of the major semiaxis is *t* counterclockwise from the X axis.

Each of the `mllib_GraphicsDrawEllipse_X_*`() functions draws an ellipse in Xor mode as follows:

$$\text{data}[x,y] \wedge= c \wedge c2$$

Each of the `mLib_GraphicsDrawEllipse_A_*`() functions draws an ellipse with antialiasing.

Each of the `mLib_GraphicsDrawEllipse_B_*`() functions draws an ellipse with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - \alpha) + c * \alpha) / 255$$

Each of the `mLib_GraphicsDrawEllipse_A_*`() functions draws an ellipse with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>x</i>	X coordinate of the center.
<i>y</i>	Y coordinate of the center.
<i>a</i>	Major semiaxis of the ellipse.
<i>b</i>	Minor semiaxis of the ellipse.
<i>t</i>	Angle of major semiaxis in radians.
<i>c</i>	Color used in the drawing.
<i>c2</i>	Alternation color.
<i>alpha</i>	Alpha value for blending. $0 \leq \alpha \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mLib_GraphicsDrawArc\(3MLIB\)](#), [mLib_GraphicsDrawCircle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawLine, mllib_GraphicsDrawLine_8, mllib_GraphicsDrawLine_32, mllib_GraphicsDrawLine_X_8, mllib_GraphicsDrawLine_X_32, mllib_GraphicsDrawLine_A_8, mllib_GraphicsDrawLine_A_32, mllib_GraphicsDrawLine_B_8, mllib_GraphicsDrawLine_B_32, mllib_GraphicsDrawLine_G_8, mllib_GraphicsDrawLine_G_32, mllib_GraphicsDrawLine_Z_8, mllib_GraphicsDrawLine_Z_32, mllib_GraphicsDrawLine_AB_8, mllib_GraphicsDrawLine_AB_32, mllib_GraphicsDrawLine_ABG_8, mllib_GraphicsDrawLine_ABG_32, mllib_GraphicsDrawLine_ABGZ_8, mllib_GraphicsDrawLine_ABGZ_32, mllib_GraphicsDrawLine_ABZ_8, mllib_GraphicsDrawLine_ABZ_32, mllib_GraphicsDrawLine_AG_8, mllib_GraphicsDrawLine_AG_32, mllib_GraphicsDrawLine_AGZ_8, mllib_GraphicsDrawLine_AGZ_32, mllib_GraphicsDrawLine_AZ_8, mllib_GraphicsDrawLine_AZ_32, mllib_GraphicsDrawLine_BG_8, mllib_GraphicsDrawLine_BG_32, mllib_GraphicsDrawLine_BGZ_8, mllib_GraphicsDrawLine_BGZ_32, mllib_GraphicsDrawLine_BZ_8, mllib_GraphicsDrawLine_BZ_32, mllib_GraphicsDrawLine_GZ_8, mllib_GraphicsDrawLine_GZ_32 – draw line

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawLine_8(mllib_image *buffer, mllib_s16 x1,
                                     mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c);

mllib_status mllib_GraphicsDrawLine_32(mllib_image *buffer, mllib_s16 x1,
                                       mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c);

mllib_status mllib_GraphicsDrawLine_X_8(mllib_image *buffer, mllib_s16 x1,
                                       mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c, mllib_s32 c2);

mllib_status mllib_GraphicsDrawLine_X_32(mllib_image *buffer, mllib_s16 x1,
                                         mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c, mllib_s32 c2);

mllib_status mllib_GraphicsDrawLine_A_8(mllib_image *buffer, mllib_s16 x1,
                                       mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c);

mllib_status mllib_GraphicsDrawLine_A_32(mllib_image *buffer, mllib_s16 x1,
                                         mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c);

mllib_status mllib_GraphicsDrawLine_B_8(mllib_image *buffer, mllib_s16 x1,
                                       mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsDrawLine_B_32(mllib_image *buffer, mllib_s16 x1,
                                         mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsDrawLine_G_8(mllib_image *buffer, mllib_s16 x1,
                                       mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c1, mllib_s32 c2);

mllib_status mllib_GraphicsDrawLine_G_32(mllib_image *buffer, mllib_s16 x1,
                                         mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s32 c1, mllib_s32 c2);
```

```

mlib_status mlib_GraphicsDrawLine_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_AB_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_AB_32(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABG_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2,
    mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABG_32(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2,
    mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_AG_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2);

mlib_status mlib_GraphicsDrawLine_AG_32(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2);

mlib_status mlib_GraphicsDrawLine_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2);

```

```
mllib_status mllib_GraphicsDrawLine_AGZ_32(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c1, mlib_s32 c2);

mllib_status mllib_GraphicsDrawLine_AZ_8(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c);

mllib_status mllib_GraphicsDrawLine_AZ_32(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c);

mllib_status mllib_GraphicsDrawLine_BG_8(mlib_image *buffer, mlib_s16 x1,
      mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_BG_32(mlib_image *buffer, mlib_s16 x1,
      mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_BGZ_8(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_BGZ_32(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_BZ_8(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_BZ_32(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLine_GZ_8(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c1, mlib_s32 c2);

mllib_status mllib_GraphicsDrawLine_GZ_32(mlib_image *buffer,
      mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
      mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
      mlib_s32 c1, mlib_s32 c2);
```

Description Each of the `mllib_GraphicsDrawLine_*`() functions draws a line between (x1,y1) and (x2,y2).

Each of the `mllib_GraphicsDrawLine_X_*`() functions draws a line between (x1,y1) and (x2,y2) in Xor mode as follows:

$$\text{data}[x,y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawLine_A_*`() functions draws a line between (x1,y1) and (x2,y2) with antialiasing.

Each of the `mllib_GraphicsDrawLine_B_*`() functions draws a line between (x1,y1) and (x2,y2) with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawLine_G_*`() functions draws a line between (x1,y1) and (x2,y2) with Gouraud shading.

Each of the `mllib_GraphicsDrawLine_Z_*`() functions draws a line between (x1,y1) and (x2,y2) with Z buffering.

Each of the other functions draws a line between (x1,y1) and (x2,y2) with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>zbuffer</i>	Pointer to the image that holds the Z buffer.
<i>x1</i>	X coordinate of the first point.
<i>y1</i>	Y coordinate of the first point.
<i>z1</i>	Z coordinate of the first point.
<i>x2</i>	X coordinate of the second point.
<i>y2</i>	Y coordinate of the second point.
<i>z2</i>	Z coordinate of the second point.
<i>c</i>	Color used in the drawing.
<i>c1</i>	Color of the first point.
<i>c2</i>	Color of the second point, or the alternation color in the Xor mode.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawLineSet\(3MLIB\)](#), [mllib_GraphicsDrawLineFanSet\(3MLIB\)](#), [mllib_GraphicsDrawLineStripSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawLineFanSet, mllib_GraphicsDrawLineFanSet_8, mllib_GraphicsDrawLineFanSet_32, mllib_GraphicsDrawLineFanSet_X_8, mllib_GraphicsDrawLineFanSet_X_32, mllib_GraphicsDrawLineFanSet_A_8, mllib_GraphicsDrawLineFanSet_A_32, mllib_GraphicsDrawLineFanSet_B_8, mllib_GraphicsDrawLineFanSet_B_32, mllib_GraphicsDrawLineFanSet_G_8, mllib_GraphicsDrawLineFanSet_G_32, mllib_GraphicsDrawLineFanSet_Z_8, mllib_GraphicsDrawLineFanSet_Z_32, mllib_GraphicsDrawLineFanSet_AB_8, mllib_GraphicsDrawLineFanSet_AB_32, mllib_GraphicsDrawLineFanSet_ABG_8, mllib_GraphicsDrawLineFanSet_ABG_32, mllib_GraphicsDrawLineFanSet_ABGZ_8, mllib_GraphicsDrawLineFanSet_ABGZ_32, mllib_GraphicsDrawLineFanSet_ABZ_8, mllib_GraphicsDrawLineFanSet_ABZ_32, mllib_GraphicsDrawLineFanSet_AG_8, mllib_GraphicsDrawLineFanSet_AG_32, mllib_GraphicsDrawLineFanSet_AGZ_8, mllib_GraphicsDrawLineFanSet_AGZ_32, mllib_GraphicsDrawLineFanSet_AZ_8, mllib_GraphicsDrawLineFanSet_AZ_32, mllib_GraphicsDrawLineFanSet_BG_8, mllib_GraphicsDrawLineFanSet_BG_32, mllib_GraphicsDrawLineFanSet_BGZ_8, mllib_GraphicsDrawLineFanSet_BGZ_32, mllib_GraphicsDrawLineFanSet_BZ_8, mllib_GraphicsDrawLineFanSet_BZ_32, mllib_GraphicsDrawLineFanSet_GZ_8, mllib_GraphicsDrawLineFanSet_GZ_32 – draw line set where all members of the set have one common end point

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```

mllib_status mllib_GraphicsDrawLineFanSet_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_X_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsDrawLineFanSet_X_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsDrawLineFanSet_A_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_A_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_B_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_B_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

```

```
mllib_status mllib_GraphicsDrawLineFanSet_G_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_G_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_Z_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_Z_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_AB_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_AB_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABG_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABG_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABGZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_ABZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_AG_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_AG_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);
```

```

mllib_status mllib_GraphicsDrawLineFanSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineFanSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineFanSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineFanSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsDrawLineFanSet_*`() functions draws a set of lines connecting (x1,y1) with (x2,y2), (x1,y1) with (x3,y3), ..., and (x1,y1) with (xn,yn).

Each of the `mllib_GraphicsDrawLineFanSet_X_*`() functions draws a set of lines in Xor mode as follows:

`data[x,y] ^= c ^ c2`

Each of the `mllib_GraphicsDrawLineFanSet_A_*`() functions draws a set of lines with antialiasing.

Each of the `mllib_GraphicsDrawLineFanSet_B_*`() functions draws a set of lines with alpha blending as follows:

`data[x,y] = (data[x,y] * (255 - a) + c * a) / 255`

Each of the `mllib_GraphicsDrawLineFanSet_G_*`() functions draws a set of lines with Gouraud shading.

Each of the `mllib_GraphicsDrawLineFanSet_Z_*`() functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x* Pointer to array of X coordinates of the points.
- y* Pointer to array of Y coordinates of the points.
- z* Pointer to array of Z coordinates of the points.
- npoints* Number of points in the arrays.
- c* Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_GraphicsDrawLine(3MLIB)`, `mlib_GraphicsDrawLineSet(3MLIB)`,
`mlib_GraphicsDrawLineStripSet(3MLIB)`, `attributes(5)`

Name mlib_GraphicsDrawLineSet, mlib_GraphicsDrawLineSet_8, mlib_GraphicsDrawLineSet_32, mlib_GraphicsDrawLineSet_X_8, mlib_GraphicsDrawLineSet_X_32, mlib_GraphicsDrawLineSet_A_8, mlib_GraphicsDrawLineSet_A_32, mlib_GraphicsDrawLineSet_B_8, mlib_GraphicsDrawLineSet_B_32, mlib_GraphicsDrawLineSet_G_8, mlib_GraphicsDrawLineSet_G_32, mlib_GraphicsDrawLineSet_Z_8, mlib_GraphicsDrawLineSet_Z_32, mlib_GraphicsDrawLineSet_AB_8, mlib_GraphicsDrawLineSet_AB_32, mlib_GraphicsDrawLineSet_ABG_8, mlib_GraphicsDrawLineSet_ABG_32, mlib_GraphicsDrawLineSet_ABGZ_8, mlib_GraphicsDrawLineSet_ABGZ_32, mlib_GraphicsDrawLineSet_ABZ_8, mlib_GraphicsDrawLineSet_ABZ_32, mlib_GraphicsDrawLineSet_AG_8, mlib_GraphicsDrawLineSet_AG_32, mlib_GraphicsDrawLineSet_AGZ_8, mlib_GraphicsDrawLineSet_AGZ_32, mlib_GraphicsDrawLineSet_AZ_8, mlib_GraphicsDrawLineSet_AZ_32, mlib_GraphicsDrawLineSet_BG_8, mlib_GraphicsDrawLineSet_BG_32, mlib_GraphicsDrawLineSet_BGZ_8, mlib_GraphicsDrawLineSet_BGZ_32, mlib_GraphicsDrawLineSet_BZ_8, mlib_GraphicsDrawLineSet_BZ_32, mlib_GraphicsDrawLineSet_GZ_8, mlib_GraphicsDrawLineSet_GZ_32 – draw line set where each member can have different end points

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mlib_GraphicsDrawLineSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mlib_GraphicsDrawLineSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mlib_GraphicsDrawLineSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mllib_status mlib_GraphicsDrawLineSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mllib_status mlib_GraphicsDrawLineSet_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mlib_GraphicsDrawLineSet_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mlib_GraphicsDrawLineSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mllib_status mlib_GraphicsDrawLineSet_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);
```

```

mllib_status mllib_GraphicsDrawLineSet_G_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_G_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_Z_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineSet_Z_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineSet_AB_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_AB_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABG_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABG_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABGZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABGZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_ABZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_AG_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

```

```
mllib_status mllib_GraphicsDrawLineSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
```


Description Each of the `mllib_GraphicsDrawLineSet_*()` functions draws a set of lines connecting (x_1, y_1) with (x_2, y_2) , (x_3, y_3) with (x_4, y_4) , ..., and (x_{n-1}, y_{n-1}) with (x_n, y_n) .

Each of the `mllib_GraphicsDrawLineSet_X_*()` functions draws a set of lines in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawLineSet_A_*()` functions draws a set of lines with antialiasing.

Each of the `mllib_GraphicsDrawLineSet_B_*()` functions draws a set of lines with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawLineSet_G_*()` functions draws a set of lines with Gouraud shading.

Each of the `mllib_GraphicsDrawLineSet_Z_*()` functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>zbuffer</i>	Pointer to the image that holds the Z buffer.
<i>x</i>	Pointer to array of X coordinates of the points.
<i>y</i>	Pointer to array of Y coordinates of the points.
<i>z</i>	Pointer to array of Z coordinates of the points.
<i>npoints</i>	Number of points in the arrays. <i>npoints</i> must be a multiple of 2.
<i>c</i>	Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
<i>c2</i>	Alternation color.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawLine\(3MLIB\)](#), [mllib_GraphicsDrawLineFanSet\(3MLIB\)](#),
[mllib_GraphicsDrawLineStripSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawLineStripSet, mllib_GraphicsDrawLineStripSet_8, mllib_GraphicsDrawLineStripSet_32, mllib_GraphicsDrawLineStripSet_X_8, mllib_GraphicsDrawLineStripSet_X_32, mllib_GraphicsDrawLineStripSet_A_8, mllib_GraphicsDrawLineStripSet_A_32, mllib_GraphicsDrawLineStripSet_B_8, mllib_GraphicsDrawLineStripSet_B_32, mllib_GraphicsDrawLineStripSet_G_8, mllib_GraphicsDrawLineStripSet_G_32, mllib_GraphicsDrawLineStripSet_Z_8, mllib_GraphicsDrawLineStripSet_Z_32, mllib_GraphicsDrawLineStripSet_AB_8, mllib_GraphicsDrawLineStripSet_AB_32, mllib_GraphicsDrawLineStripSet_ABG_8, mllib_GraphicsDrawLineStripSet_ABG_32, mllib_GraphicsDrawLineStripSet_ABGZ_8, mllib_GraphicsDrawLineStripSet_ABGZ_32, mllib_GraphicsDrawLineStripSet_ABZ_8, mllib_GraphicsDrawLineStripSet_ABZ_32, mllib_GraphicsDrawLineStripSet_AG_8, mllib_GraphicsDrawLineStripSet_AG_32, mllib_GraphicsDrawLineStripSet_AGZ_8, mllib_GraphicsDrawLineStripSet_AGZ_32, mllib_GraphicsDrawLineStripSet_AZ_8, mllib_GraphicsDrawLineStripSet_AZ_32, mllib_GraphicsDrawLineStripSet_BG_8, mllib_GraphicsDrawLineStripSet_BG_32, mllib_GraphicsDrawLineStripSet_BGZ_8, mllib_GraphicsDrawLineStripSet_BGZ_32, mllib_GraphicsDrawLineStripSet_BZ_8, mllib_GraphicsDrawLineStripSet_BZ_32, mllib_GraphicsDrawLineStripSet_GZ_8, mllib_GraphicsDrawLineStripSet_GZ_32 – draw line set where each member of the set starts at the point where the previous member ended

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```

mllib_status mllib_GraphicsDrawLineStripSet_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_X_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 c2);

mllib_status mllib_GraphicsDrawLineStripSet_X_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 c2);

mllib_status mllib_GraphicsDrawLineStripSet_A_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_A_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_B_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_B_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

```

```
mlib_status mlib_GraphicsDrawLineStripSet_G_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineStripSet_G_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineStripSet_Z_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineStripSet_Z_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z,
        mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineStripSet_AB_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_AB_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABG_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABGZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABGZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_AG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);
```

```

mllib_status mllib_GraphicsDrawLineStripSet_AG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineStripSet_AGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineStripSet_AGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineStripSet_AZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_AZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawLineStripSet_BG_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_BG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_BGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_BGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_BZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_BZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawLineStripSet_GZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawLineStripSet_GZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z,
    mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsDrawLineStripSet_*`() functions draws a set of lines connecting (x1,y1) with (x2,y2), (x2,y2) with (x3,y3), ..., and (xn-1,yn-1) with (xn,yn).

Each of the `mllib_GraphicsDrawLineStripSet_X_*`() functions draws a set of lines in Xor mode as follows:

$$\text{data}[x,y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawLineStripSet_A_*`() functions draws a set of lines with antialiasing.

Each of the `mllib_GraphicsDrawLineStripSet_B_*`() functions draws a set of lines with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawLineStripSet_G_*`() functions draws a set of lines with Gouraud shading.

Each of the `mllib_GraphicsDrawLineStripSet_Z_*`() functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Note that the `mllib_GraphicsDrawPolyline_*`() functions are aliases of the `mllib_GraphicsDrawLineStripSet_*`() functions.

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x Pointer to array of X coordinates of the points.

y Pointer to array of Y coordinates of the points.

z Pointer to array of Z coordinates of the points.

npoints Number of points in the arrays.

c Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawLine\(3MLIB\)](#), [mllib_GraphicsDrawLineSet\(3MLIB\)](#),
[mllib_GraphicsDrawLineFanSet\(3MLIB\)](#), [mllib_GraphicsDrawPolyline\(3MLIB\)](#),
[attributes\(5\)](#)

Name mllib_GraphicsDrawPoint, mllib_GraphicsDrawPoint_8, mllib_GraphicsDrawPoint_32, mllib_GraphicsDrawPoint_X_8, mllib_GraphicsDrawPoint_X_32, mllib_GraphicsDrawPoint_B_8, mllib_GraphicsDrawPoint_B_32 – draw point

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawPoint_8(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawPoint_32(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawPoint_X_8(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c, mllib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPoint_X_32(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c, mllib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPoint_B_8(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c, mllib_s32 a);  
  
mllib_status mllib_GraphicsDrawPoint_B_32(mllib_image *buffer, mllib_s16 x,  
                                       mllib_s16 y, mllib_s32 c, mllib_s32 a);
```

Description Each of the `mllib_GraphicsDrawPoint_*`() functions draws a point at (*x*,*y*) in color *c*.

Each of the `mllib_GraphicsDrawPoint_X_*`() functions draws a point at (*x*,*y*) in Xor mode as follows:

$$\text{data}[\textit{x}, \textit{y}] \hat{=} c \hat{ } c2$$

Each of the `mllib_GraphicsDrawPoint_B_*`() functions draws a point at (*x*,*y*) with alpha blending as follows:

$$\text{data}[\textit{x}, \textit{y}] = (\text{data}[\textit{x}, \textit{y}] * (255 - a) + c * a) / 255$$

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.
x X coordinate of the point.
y Y coordinate of the point.
c Color used in the drawing.
c2 Alternation color.
a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawPointSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawPointSet, mllib_GraphicsDrawPointSet_8,
mllib_GraphicsDrawPointSet_32, mllib_GraphicsDrawPointSet_X_8,
mllib_GraphicsDrawPointSet_X_32, mllib_GraphicsDrawPointSet_B_8,
mllib_GraphicsDrawPointSet_B_32, mllib_GraphicsDrawPolypoint_8,
mllib_GraphicsDrawPolypoint_32, mllib_GraphicsDrawPolypoint_X_8,
mllib_GraphicsDrawPolypoint_X_32, mllib_GraphicsDrawPolypoint_B_8,
mllib_GraphicsDrawPolypoint_B_32 – draw a set of points

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawPointSet_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPointSet_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPointSet_X_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPointSet_X_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPointSet_B_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawPointSet_B_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawPolypoint_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolypoint_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolypoint_X_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPolypoint_X_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPolypoint_B_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPolypoint_B_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);
```

Description Each of the `mlib_GraphicsDrawPointSet_*`() and `mlib_GraphicsDrawPolypoint_*`() functions draws a set of points at (x_1, y_1) , (x_2, y_2) , ..., and (x_n, y_n) .

Each of the `mlib_GraphicsDrawPointSet_X_*`() and `mlib_GraphicsDrawPolypoint_X_*`() functions draws a set of points at (x_1, y_1) , (x_2, y_2) , ..., and (x_n, y_n) in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mlib_GraphicsDrawPointSet_B_*`() and `mlib_GraphicsDrawPolypoint_B_*`() functions draws a set of points at (x_1, y_1) , (x_2, y_2) , ..., and (x_n, y_n) with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

The `mlib_GraphicsDrawPolypoint_*`() functions are aliases of the `mlib_GraphicsDrawPointSet_*`() functions.

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

x Pointer to array of X coordinates of the points.

y Pointer to array of Y coordinates of the points.

npoints Number of points in the arrays.

c Color used in the drawing.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_GraphicsDrawPoint\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawPolygon, mllib_GraphicsDrawPolygon_8,
mllib_GraphicsDrawPolygon_32, mllib_GraphicsDrawPolygon_X_8,
mllib_GraphicsDrawPolygon_X_32, mllib_GraphicsDrawPolygon_A_8,
mllib_GraphicsDrawPolygon_A_32, mllib_GraphicsDrawPolygon_B_8,
mllib_GraphicsDrawPolygon_B_32, mllib_GraphicsDrawPolygon_G_8,
mllib_GraphicsDrawPolygon_G_32, mllib_GraphicsDrawPolygon_Z_8,
mllib_GraphicsDrawPolygon_Z_32, mllib_GraphicsDrawPolygon_AB_8,
mllib_GraphicsDrawPolygon_AB_32, mllib_GraphicsDrawPolygon_ABG_8,
mllib_GraphicsDrawPolygon_ABG_32, mllib_GraphicsDrawPolygon_ABGZ_8,
mllib_GraphicsDrawPolygon_ABGZ_32, mllib_GraphicsDrawPolygon_ABZ_8,
mllib_GraphicsDrawPolygon_ABZ_32, mllib_GraphicsDrawPolygon_AG_8,
mllib_GraphicsDrawPolygon_AG_32, mllib_GraphicsDrawPolygon_AGZ_8,
mllib_GraphicsDrawPolygon_AGZ_32, mllib_GraphicsDrawPolygon_AZ_8,
mllib_GraphicsDrawPolygon_AZ_32, mllib_GraphicsDrawPolygon_BG_8,
mllib_GraphicsDrawPolygon_BG_32, mllib_GraphicsDrawPolygon_BGZ_8,
mllib_GraphicsDrawPolygon_BGZ_32, mllib_GraphicsDrawPolygon_BZ_8,
mllib_GraphicsDrawPolygon_BZ_32, mllib_GraphicsDrawPolygon_GZ_8,
mllib_GraphicsDrawPolygon_GZ_32 – draw polygon

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawPolygon_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolygon_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolygon_X_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPolygon_X_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawPolygon_A_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolygon_A_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawPolygon_B_8(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawPolygon_B_32(mllib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,  
    mlib_s32 a);
```

```

mllib_status mllib_GraphicsDrawPolygon_G_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolygon_G_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y,
      mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolygon_Z_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolygon_Z_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolygon_AB_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y,
      mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_AB_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABG_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABGZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABGZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_ABZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_AG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

```

```
mllib_status mllib_GraphicsDrawPolygon_AG_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolygon_AGZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolygon_AGZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolygon_AZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c);

mllib_status mllib_GraphicsDrawPolygon_AZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c);

mllib_status mllib_GraphicsDrawPolygon_BG_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_BG_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_BGZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_BGZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_BZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_BZ_32(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolygon_GZ_8(mlib_image *buffer,
      mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      const mlib_s32 *c);
```

```
mLib_status mLib_GraphicsDrawPolygon_GZ_32(mLib_image *buffer,
      mLib_image *zbuffer, const mLib_s16 *x, const mLib_s16 *y,
      const mLib_s16 *z, mLib_s32 npoints,
      const mLib_s32 *c);
```

Description Each of the `mLib_GraphicsDrawPolygon_*()` functions draws a polygon enclosing (x_1, y_1) , (x_2, y_2) , ..., and (x_n, y_n) .

Each of the `mLib_GraphicsDrawPolygon_X_*()` functions draws a polygon in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mLib_GraphicsDrawPolygon_A_*()` functions draws a polygon with antialiasing.

Each of the `mLib_GraphicsDrawPolygon_B_*()` functions draws a polygon with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mLib_GraphicsDrawPolygon_G_*()` functions draws a polygon with Gouraud shading.

Each of the `mLib_GraphicsDrawPolygon_Z_*()` functions draws a polygon with Z buffering.

Each of the other functions draws a polygon with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>zbuffer</i>	Pointer to the image that holds the Z buffer.
<i>x</i>	Pointer to the array of X coordinates of the vertices.
<i>y</i>	Pointer to the array of Y coordinates of the vertices.
<i>z</i>	Pointer to the array of Z coordinates of the vertices.
<i>npoints</i>	Number of vertices in the arrays.
<i>c</i>	Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
<i>c2</i>	Alternation color.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillPolygon\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawPolyline, mllib_GraphicsDrawPolyline_8,
 mllib_GraphicsDrawPolyline_32, mllib_GraphicsDrawPolyline_X_8,
 mllib_GraphicsDrawPolyline_X_32, mllib_GraphicsDrawPolyline_A_8,
 mllib_GraphicsDrawPolyline_A_32, mllib_GraphicsDrawPolyline_B_8,
 mllib_GraphicsDrawPolyline_B_32, mllib_GraphicsDrawPolyline_G_8,
 mllib_GraphicsDrawPolyline_G_32, mllib_GraphicsDrawPolyline_Z_8,
 mllib_GraphicsDrawPolyline_Z_32, mllib_GraphicsDrawPolyline_AB_8,
 mllib_GraphicsDrawPolyline_AB_32, mllib_GraphicsDrawPolyline_ABG_8,
 mllib_GraphicsDrawPolyline_ABG_32, mllib_GraphicsDrawPolyline_ABGZ_8,
 mllib_GraphicsDrawPolyline_ABGZ_32, mllib_GraphicsDrawPolyline_ABZ_8,
 mllib_GraphicsDrawPolyline_ABZ_32, mllib_GraphicsDrawPolyline_AG_8,
 mllib_GraphicsDrawPolyline_AG_32, mllib_GraphicsDrawPolyline_AGZ_8,
 mllib_GraphicsDrawPolyline_AGZ_32, mllib_GraphicsDrawPolyline_AZ_8,
 mllib_GraphicsDrawPolyline_AZ_32, mllib_GraphicsDrawPolyline_BG_8,
 mllib_GraphicsDrawPolyline_BG_32, mllib_GraphicsDrawPolyline_BGZ_8,
 mllib_GraphicsDrawPolyline_BGZ_32, mllib_GraphicsDrawPolyline_BZ_8,
 mllib_GraphicsDrawPolyline_BZ_32, mllib_GraphicsDrawPolyline_GZ_8,
 mllib_GraphicsDrawPolyline_GZ_32 – draw polyline

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```

mllib_status mllib_GraphicsDrawPolyline_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_X_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsDrawPolyline_X_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsDrawPolyline_A_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_A_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_B_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_B_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

```

```
mllib_status mllib_GraphicsDrawPolyline_G_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_G_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_Z_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_Z_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_AB_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_AB_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABG_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABGZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABGZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_ABZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_AG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_AG_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);
```

```

mllib_status mllib_GraphicsDrawPolyline_AGZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_AGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_AZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_AZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawPolyline_BG_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_BG_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_BGZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_BGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_BZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_BZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawPolyline_GZ_8(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawPolyline_GZ_32(mllib_image *buffer,
        mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsDrawPolyline_*`() functions draws a polyline connecting (x1,y1), (x2,y2), ..., and (xn,yn).

Each of the `mllib_GraphicsDrawPolyline_X_*`() functions draws a polyline in Xor mode as follows:

```
data[x,y] ^= c ^ c2
```

Each of the `mllib_GraphicsDrawPolyline_A_*`() functions draws a polyline with antialiasing.

Each of the `mllib_GraphicsDrawPolyline_B_*`() functions draws a polyline with alpha blending as follows:

```
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
```

Each of the `mllib_GraphicsDrawPolyline_G_*`() functions draws a polyline with Gouraud shading.

Each of the `mllib_GraphicsDrawPolyline_Z_*`() functions draws a polyline with Z buffering.

Each of the other functions draws a polyline with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Note that the `mllib_GraphicsDrawPolyline_*`() functions are aliases of the `mllib_GraphicsDrawLineStripSet_*`() functions.

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x* Pointer to array of X coordinates of the points.
- y* Pointer to array of Y coordinates of the points.
- z* Pointer to array of Z coordinates of the points.
- npoints* Number of points in the arrays.
- c* Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_GraphicsDrawLine(3MLIB)`, `mlib_GraphicsDrawLineSet(3MLIB)`,
`mlib_GraphicsDrawLineFanSet(3MLIB)`, `mlib_GraphicsDrawLineStripSet(3MLIB)`,
`attributes(5)`

Name mllib_GraphicsDrawRectangle, mllib_GraphicsDrawRectangle_8,
mllib_GraphicsDrawRectangle_32, mllib_GraphicsDrawRectangle_X_8,
mllib_GraphicsDrawRectangle_X_32, mllib_GraphicsDrawRectangle_B_8,
mllib_GraphicsDrawRectangle_B_32 – draw rectangle

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_GraphicsDrawRectangle_8(mllib_image *buffer, mlib_s16 x,  
      mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawRectangle_32(mllib_image *buffer, mlib_s16 x,  
      mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);  
  
mllib_status mllib_GraphicsDrawRectangle_X_8(mllib_image *buffer, mlib_s16 x,  
      mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawRectangle_X_32(mllib_image *buffer,  
      mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c,  
      mlib_s32 c2);  
  
mllib_status mllib_GraphicsDrawRectangle_B_8(mllib_image *buffer, mlib_s16 x,  
      mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsDrawRectangle_B_32(mllib_image *buffer,  
      mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c,  
      mlib_s32 a);
```

Description Each of the `mllib_GraphicsDrawRectangle_*`() functions draws a rectangle with the upper-left corner at (x, y), width w, and height h.

Each of the `mllib_GraphicsDrawRectangle_X_*`() functions draws a rectangle in Xor mode as follows:

$$\text{data}[x,y] \hat{=} c \hat{^} c2$$

Each of the `mllib_GraphicsDrawRectangle_B_*`() functions draws a rectangle with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.
x X coordinate of the upper-left corner of the rectangle.
y Y coordinate of the upper-left corner of the rectangle.
w Width of the rectangle.
h Height of the rectangle.
c Color used in the drawing.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillRectangle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawTriangle, mllib_GraphicsDrawTriangle_8,
mllib_GraphicsDrawTriangle_32, mllib_GraphicsDrawTriangle_X_8,
mllib_GraphicsDrawTriangle_X_32, mllib_GraphicsDrawTriangle_A_8,
mllib_GraphicsDrawTriangle_A_32, mllib_GraphicsDrawTriangle_B_8,
mllib_GraphicsDrawTriangle_B_32, mllib_GraphicsDrawTriangle_G_8,
mllib_GraphicsDrawTriangle_G_32, mllib_GraphicsDrawTriangle_Z_8,
mllib_GraphicsDrawTriangle_Z_32, mllib_GraphicsDrawTriangle_AB_8,
mllib_GraphicsDrawTriangle_AB_32, mllib_GraphicsDrawTriangle_ABG_8,
mllib_GraphicsDrawTriangle_ABG_32, mllib_GraphicsDrawTriangle_ABGZ_8,
mllib_GraphicsDrawTriangle_ABGZ_32, mllib_GraphicsDrawTriangle_ABZ_8,
mllib_GraphicsDrawTriangle_ABZ_32, mllib_GraphicsDrawTriangle_AG_8,
mllib_GraphicsDrawTriangle_AG_32, mllib_GraphicsDrawTriangle_AGZ_8,
mllib_GraphicsDrawTriangle_AGZ_32, mllib_GraphicsDrawTriangle_AZ_8,
mllib_GraphicsDrawTriangle_AZ_32, mllib_GraphicsDrawTriangle_BG_8,
mllib_GraphicsDrawTriangle_BG_32, mllib_GraphicsDrawTriangle_BGZ_8,
mllib_GraphicsDrawTriangle_BGZ_32, mllib_GraphicsDrawTriangle_BZ_8,
mllib_GraphicsDrawTriangle_BZ_32, mllib_GraphicsDrawTriangle_GZ_8,
mllib_GraphicsDrawTriangle_GZ_32 – draw triangle

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawTriangle_8(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawTriangle_32(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawTriangle_X_8(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c, mllib_s32 c2);  
  
mllib_status mllib_GraphicsDrawTriangle_X_32(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c, mllib_s32 c2);  
  
mllib_status mllib_GraphicsDrawTriangle_A_8(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawTriangle_A_32(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c);  
  
mllib_status mllib_GraphicsDrawTriangle_B_8(mllib_image *buffer, mllib_s16 x1,  
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,  
    mllib_s32 c, mllib_s32 a);
```



```

mllib_status mllib_GraphicsDrawTriangle_B_32(mllib_image *buffer, mllib_s16 x1,
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
    mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_G_8(mllib_image *buffer, mllib_s16 x1,
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
    mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_G_32(mllib_image *buffer, mllib_s16 x1,
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
    mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_Z_8(mllib_image *buffer,
    mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
    mllib_s16 x2, mllib_s16 y2, mllib_s16 z2,
    mllib_s16 x3, mllib_s16 y3, mllib_s16 z3, mllib_s32 c);

mllib_status mllib_GraphicsDrawTriangle_Z_32(mllib_image *buffer,
    mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
    mllib_s16 x2, mllib_s16 y2, mllib_s16 z2,
    mllib_s16 x3, mllib_s16 y3, mllib_s16 z3, mllib_s32 c);

mllib_status mllib_GraphicsDrawTriangle_AB_8(mllib_image *buffer, mllib_s16 x1,
    mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
    mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_AB_32(mllib_image *buffer,
    mllib_s16 x1, mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3,
    mllib_s16 y3, mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABG_8(mllib_image *buffer,
    mllib_s16 x1, mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3,
    mllib_s16 y3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABG_32(mllib_image *buffer,
    mllib_s16 x1, mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3,
    mllib_s16 y3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
    mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
    mllib_s16 z3, mllib_s32 c1,
    mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
    mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
    mllib_s16 z3, mllib_s32 c1,
    mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABZ_8(mllib_image *buffer,
    mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
    mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,

```

```
    mlib_s16 z3, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_AG_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_AG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1,
    mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1,
    mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangle_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangle_BG_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_BG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1,
    mlib_s32 c2, mlib_s32 c3, mlib_s32 a);
```

```

mllib_status mllib_GraphicsDrawTriangle_BGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c1,
        mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_BZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
        mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_BZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangle_GZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
        mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsDrawTriangle_GZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c1,
        mlib_s32 c2, mlib_s32 c3);

```

Description Each of the `mllib_GraphicsDrawTriangle_*()` functions draws a triangle with the vertices at $(x1, y1)$, $(x2, y2)$, and $(x3, y3)$.

Each of the `mllib_GraphicsDrawTriangle_X_*()` functions draws a triangle in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawTriangle_A_*()` functions draws a triangle with antialiasing.

Each of the `mllib_GraphicsDrawTriangle_B_*()` functions draws a triangle with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawTriangle_G_*()` functions draws a triangle with Gouraud shading.

Each of the `mllib_GraphicsDrawTriangle_Z_*()` functions draws a triangle with Z buffering.

Each of the other functions draws a triangle with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x1* X coordinate of the first vertex.
- y1* Y coordinate of the first vertex.
- z1* Z coordinate of the first vertex.
- x2* X coordinate of the second vertex.
- y2* Y coordinate of the second vertex.
- z2* Z coordinate of the second vertex.
- x3* X coordinate of the third vertex.
- y3* Y coordinate of the third vertex.
- z3* Z coordinate of the third vertex.
- c* Color used in the drawing.
- c1* Color of the first vertex.
- c2* Color of the second vertex, or the alternation color in Xor Mode.
- c3* Color of the third vertex.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillTriangle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawTriangleFanSet, mllib_GraphicsDrawTriangleFanSet_8, mllib_GraphicsDrawTriangleFanSet_32, mllib_GraphicsDrawTriangleFanSet_X_8, mllib_GraphicsDrawTriangleFanSet_X_32, mllib_GraphicsDrawTriangleFanSet_A_8, mllib_GraphicsDrawTriangleFanSet_A_32, mllib_GraphicsDrawTriangleFanSet_B_8, mllib_GraphicsDrawTriangleFanSet_B_32, mllib_GraphicsDrawTriangleFanSet_G_8, mllib_GraphicsDrawTriangleFanSet_G_32, mllib_GraphicsDrawTriangleFanSet_Z_8, mllib_GraphicsDrawTriangleFanSet_Z_32, mllib_GraphicsDrawTriangleFanSet_AB_8, mllib_GraphicsDrawTriangleFanSet_AB_32, mllib_GraphicsDrawTriangleFanSet_ABG_8, mllib_GraphicsDrawTriangleFanSet_ABG_32, mllib_GraphicsDrawTriangleFanSet_ABGZ_8, mllib_GraphicsDrawTriangleFanSet_ABGZ_32, mllib_GraphicsDrawTriangleFanSet_ABZ_8, mllib_GraphicsDrawTriangleFanSet_ABZ_32, mllib_GraphicsDrawTriangleFanSet_AG_8, mllib_GraphicsDrawTriangleFanSet_AG_32, mllib_GraphicsDrawTriangleFanSet_AGZ_8, mllib_GraphicsDrawTriangleFanSet_AGZ_32, mllib_GraphicsDrawTriangleFanSet_AZ_8, mllib_GraphicsDrawTriangleFanSet_AZ_32, mllib_GraphicsDrawTriangleFanSet_BG_8, mllib_GraphicsDrawTriangleFanSet_BG_32, mllib_GraphicsDrawTriangleFanSet_BGZ_8, mllib_GraphicsDrawTriangleFanSet_BGZ_32, mllib_GraphicsDrawTriangleFanSet_BZ_8, mllib_GraphicsDrawTriangleFanSet_BZ_32, mllib_GraphicsDrawTriangleFanSet_GZ_8, mllib_GraphicsDrawTriangleFanSet_GZ_32 – draw triangle set where all members of the set have a common vertex

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawTriangleFanSet_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_X_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleFanSet_X_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleFanSet_A_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_A_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_B_8(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_B_32(mllib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);
```

```
mllib_status mllib_GraphicsDrawTriangleFanSet_G_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_G_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z,
    mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z,
    mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
```

```

mllib_status mllib_GraphicsDrawTriangleFanSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleFanSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleFanSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleFanSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsDrawTriangleFanSet_*`() functions draws a set of triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, $\{(x_1, y_1), (x_3, y_3), (x_4, y_4)\}$, ..., and $\{(x_1, y_1), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mllib_GraphicsDrawTriangleFanSet_X_*`() functions draws a set of triangles in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawTriangleFanSet_A_*`() functions draws a set of triangles with antialiasing.

Each of the `mllib_GraphicsDrawTriangleFanSet_B_*`() functions draws a set of triangles with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawTriangleFanSet_G_*`() functions draws a set of triangles with Gouraud shading.

Each of the `mllib_GraphicsDrawTriangleFanSet_Z_*`() functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x Pointer to array of X coordinates of the points.

y Pointer to array of Y coordinates of the points.

z Pointer to array of Z coordinates of the points.

npoints Number of points in the arrays.

c Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_GraphicsDrawTriangle\(3MLIB\)](#), [mlib_GraphicsDrawTriangleSet\(3MLIB\)](#),
[mlib_GraphicsDrawTriangleStripSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawTriangleSet, mllib_GraphicsDrawTriangleSet_8, mllib_GraphicsDrawTriangleSet_32, mllib_GraphicsDrawTriangleSet_X_8, mllib_GraphicsDrawTriangleSet_X_32, mllib_GraphicsDrawTriangleSet_A_8, mllib_GraphicsDrawTriangleSet_A_32, mllib_GraphicsDrawTriangleSet_B_8, mllib_GraphicsDrawTriangleSet_B_32, mllib_GraphicsDrawTriangleSet_G_8, mllib_GraphicsDrawTriangleSet_G_32, mllib_GraphicsDrawTriangleSet_Z_8, mllib_GraphicsDrawTriangleSet_Z_32, mllib_GraphicsDrawTriangleSet_AB_8, mllib_GraphicsDrawTriangleSet_AB_32, mllib_GraphicsDrawTriangleSet_ABG_8, mllib_GraphicsDrawTriangleSet_ABG_32, mllib_GraphicsDrawTriangleSet_ABGZ_8, mllib_GraphicsDrawTriangleSet_ABGZ_32, mllib_GraphicsDrawTriangleSet_ABZ_8, mllib_GraphicsDrawTriangleSet_ABZ_32, mllib_GraphicsDrawTriangleSet_AG_8, mllib_GraphicsDrawTriangleSet_AG_32, mllib_GraphicsDrawTriangleSet_AGZ_8, mllib_GraphicsDrawTriangleSet_AGZ_32, mllib_GraphicsDrawTriangleSet_AZ_8, mllib_GraphicsDrawTriangleSet_AZ_32, mllib_GraphicsDrawTriangleSet_BG_8, mllib_GraphicsDrawTriangleSet_BG_32, mllib_GraphicsDrawTriangleSet_BGZ_8, mllib_GraphicsDrawTriangleSet_BGZ_32, mllib_GraphicsDrawTriangleSet_BZ_8, mllib_GraphicsDrawTriangleSet_BZ_32, mllib_GraphicsDrawTriangleSet_GZ_8, mllib_GraphicsDrawTriangleSet_GZ_32 – draw triangle set where each member can have different vertices

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawTriangleSet_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_X_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleSet_X_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleSet_A_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_A_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_B_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_B_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);
```

```

mllib_status mllib_GraphicsDrawTriangleSet_G_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_G_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_Z_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_Z_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_AB_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_AB_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABG_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABGZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABGZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_ABZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_AG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_AG_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

```

```
mllib_status mllib_GraphicsDrawTriangleSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
```

Description Each of the `mllib_GraphicsDrawTriangleSet_*`() functions draws a set of triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}, \{(x_4, y_4), (x_5, y_5), (x_6, y_6)\}, \dots$, and $\{(x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mllib_GraphicsDrawTriangleSet_X_*`() functions draws a set of triangles in Xor mode as follows:

$$\text{data}[x,y] \hat{=} c \wedge c2$$

Each of the `mllib_GraphicsDrawTriangleSet_A_*` functions draws a set of triangles with antialiasing.

Each of the `mllib_GraphicsDrawTriangleSet_B_*` functions draws a set of triangles with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawTriangleSet_G_*` functions draws a set of triangles with Gouraud shading.

Each of the `mllib_GraphicsDrawTriangleSet_Z_*` functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x* Pointer to array of X coordinates of the points.
- y* Pointer to array of Y coordinates of the points.
- z* Pointer to array of Z coordinates of the points.
- npoints* Number of points in the arrays. *npoints* must be a multiple of 3.
- c* Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawTriangle\(3MLIB\)](#), [mllib_GraphicsDrawTriangleFanSet\(3MLIB\)](#),
[mllib_GraphicsDrawTriangleStripSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsDrawTriangleStripSet, mllib_GraphicsDrawTriangleStripSet_8, mllib_GraphicsDrawTriangleStripSet_32, mllib_GraphicsDrawTriangleStripSet_X_8, mllib_GraphicsDrawTriangleStripSet_X_32, mllib_GraphicsDrawTriangleStripSet_A_8, mllib_GraphicsDrawTriangleStripSet_A_32, mllib_GraphicsDrawTriangleStripSet_B_8, mllib_GraphicsDrawTriangleStripSet_B_32, mllib_GraphicsDrawTriangleStripSet_G_8, mllib_GraphicsDrawTriangleStripSet_G_32, mllib_GraphicsDrawTriangleStripSet_Z_8, mllib_GraphicsDrawTriangleStripSet_Z_32, mllib_GraphicsDrawTriangleStripSet_AB_8, mllib_GraphicsDrawTriangleStripSet_AB_32, mllib_GraphicsDrawTriangleStripSet_ABG_8, mllib_GraphicsDrawTriangleStripSet_ABG_32, mllib_GraphicsDrawTriangleStripSet_ABGZ_8, mllib_GraphicsDrawTriangleStripSet_ABGZ_32, mllib_GraphicsDrawTriangleStripSet_ABZ_8, mllib_GraphicsDrawTriangleStripSet_ABZ_32, mllib_GraphicsDrawTriangleStripSet_AG_8, mllib_GraphicsDrawTriangleStripSet_AG_32, mllib_GraphicsDrawTriangleStripSet_AGZ_8, mllib_GraphicsDrawTriangleStripSet_AGZ_32, mllib_GraphicsDrawTriangleStripSet_AZ_8, mllib_GraphicsDrawTriangleStripSet_AZ_32, mllib_GraphicsDrawTriangleStripSet_BG_8, mllib_GraphicsDrawTriangleStripSet_BG_32, mllib_GraphicsDrawTriangleStripSet_BGZ_8, mllib_GraphicsDrawTriangleStripSet_BGZ_32, mllib_GraphicsDrawTriangleStripSet_BZ_8, mllib_GraphicsDrawTriangleStripSet_BZ_32, mllib_GraphicsDrawTriangleStripSet_GZ_8, mllib_GraphicsDrawTriangleStripSet_GZ_32 – draw triangle set where the first side of each member is common to the second side of the previous member

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsDrawTriangleStripSet_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_X_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleStripSet_X_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsDrawTriangleStripSet_A_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_A_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_B_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
```

```
mllib_status mllib_GraphicsDrawTriangleStripSet_B_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_G_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_G_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_Z_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_Z_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_AB_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_AB_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABG_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_ABZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_AG_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
```



```

mllib_status mllib_GraphicsDrawTriangleStripSet_AG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_AGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_AGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_AZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_AZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsDrawTriangleStripSet_BG_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_BG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_BGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_BGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_BZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_BZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsDrawTriangleStripSet_GZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsDrawTriangleStripSet_GZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsDrawTriangleStripSet_*()` functions draws a set of triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, $\{(x_2, y_2), (x_3, y_3), (x_4, y_4)\}$, ..., and $\{(x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mllib_GraphicsDrawTriangleStripSet_X_*()` functions draws a set of triangles in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsDrawTriangleStripSet_A_*()` functions draws a set of triangles with antialiasing.

Each of the `mllib_GraphicsDrawTriangleStripSet_B_*()` functions draws a set of triangles with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsDrawTriangleStripSet_G_*()` functions draws a set of triangles with Gouraud shading.

Each of the `mllib_GraphicsDrawTriangleStripSet_Z_*()` functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x Pointer to array of X coordinates of the points.

y Pointer to array of Y coordinates of the points.

z Pointer to array of Z coordinates of the points.

npoints Number of points in the arrays.

c Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawTriangle\(3MLIB\)](#), [mllib_GraphicsDrawTriangleSet\(3MLIB\)](#),
[mllib_GraphicsDrawTriangleFanSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillArc, mllib_GraphicsFillArc_8, mllib_GraphicsFillArc_32, mllib_GraphicsFillArc_X_8, mllib_GraphicsFillArc_X_32, mllib_GraphicsFillArc_A_8, mllib_GraphicsFillArc_A_32, mllib_GraphicsFillArc_B_8, mllib_GraphicsFillArc_B_32, mllib_GraphicsFillArc_AB_8, mllib_GraphicsFillArc_AB_32 – draw filled arc

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_GraphicsFillArc_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillArc_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillArc_X_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 c2);  
  
mllib_status mllib_GraphicsFillArc_X_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 c2);  
  
mllib_status mllib_GraphicsFillArc_A_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillArc_A_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillArc_B_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 a);  
  
mllib_status mllib_GraphicsFillArc_B_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 a);  
  
mllib_status mllib_GraphicsFillArc_AB_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 a);  
  
mllib_status mllib_GraphicsFillArc_AB_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 r, mllib_f32 t1, mllib_f32 t2, mllib_s32 c,  
    mllib_s32 a);
```

Description Each of the mllib_GraphicsFillArc_*() functions draws a filled arc with the center at (x,y), radius r, start angle t1, and end angle t2.

Each of the mllib_GraphicsFillArc_X_*() functions draws a filled arc in Xor mode as follows:

```
data[x,y] ^= c ^ c2
```

Each of the mllib_GraphicsFillArc_A_*() functions draws a filled arc with antialiasing.

Each of the `mllib_GraphicsFillArc_B_*`() functions draws a filled arc with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillArc_AB_*`() functions draws a filled arc with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>x</i>	X coordinate of the center.
<i>y</i>	Y coordinate of the center.
<i>r</i>	Radius of the arc.
<i>t1</i>	Start angle of the arc in radians.
<i>t2</i>	End angle of the arc in radians.
<i>c</i>	Color used in the drawing.
<i>c2</i>	Alternation color.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillCircle\(3MLIB\)](#), [mllib_GraphicsFillEllipse\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillCircle, mllib_GraphicsFillCircle_8, mllib_GraphicsFillCircle_32, mllib_GraphicsFillCircle_X_8, mllib_GraphicsFillCircle_X_32, mllib_GraphicsFillCircle_A_8, mllib_GraphicsFillCircle_A_32, mllib_GraphicsFillCircle_B_8, mllib_GraphicsFillCircle_B_32, mllib_GraphicsFillCircle_AB_8, mllib_GraphicsFillCircle_AB_32 – draw filled circle

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_GraphicsFillCircle_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsFillCircle_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsFillCircle_X_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsFillCircle_X_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);  
  
mllib_status mllib_GraphicsFillCircle_A_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsFillCircle_A_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c);  
  
mllib_status mllib_GraphicsFillCircle_B_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsFillCircle_B_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsFillCircle_AB_8(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);  
  
mllib_status mllib_GraphicsFillCircle_AB_32(mllib_image *buffer, mlib_s16 x,  
    mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
```

Description Each of the `mllib_GraphicsFillCircle_*`() functions draws a filled circle with the center at (x, y) and radius r .

Each of the `mllib_GraphicsFillCircle_X_*`() functions draws a filled circle in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsFillCircle_A_*`() functions draws a filled circle with antialiasing.

Each of the `mllib_GraphicsFillCircle_B_*`() functions draws a filled circle with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillColor_AB_*` functions draws a filled circle with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.
x X coordinate of the center.
y Y coordinate of the center.
r Radius of the arc.
c Color used in the drawing.
c2 Alternation color.
a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillArc\(3MLIB\)](#), [mllib_GraphicsFillEllipse\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillEllipse, mllib_GraphicsFillEllipse_8, mllib_GraphicsFillEllipse_32, mllib_GraphicsFillEllipse_X_8, mllib_GraphicsFillEllipse_X_32, mllib_GraphicsFillEllipse_A_8, mllib_GraphicsFillEllipse_A_32, mllib_GraphicsFillEllipse_B_8, mllib_GraphicsFillEllipse_B_32, mllib_GraphicsFillEllipse_AB_8, mllib_GraphicsFillEllipse_AB_32 – draw filled ellipse

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_GraphicsFillEllipse_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillEllipse_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillEllipse_X_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 c2);  
  
mllib_status mllib_GraphicsFillEllipse_X_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 c2);  
  
mllib_status mllib_GraphicsFillEllipse_A_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillEllipse_A_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c);  
  
mllib_status mllib_GraphicsFillEllipse_B_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 alpha);  
  
mllib_status mllib_GraphicsFillEllipse_B_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 alpha);  
  
mllib_status mllib_GraphicsFillEllipse_AB_8(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 alpha);  
  
mllib_status mllib_GraphicsFillEllipse_AB_32(mllib_image *buffer, mllib_s16 x,  
    mllib_s16 y, mllib_s32 a, mllib_s32 b, mllib_f32 t, mllib_s32 c,  
    mllib_s32 alpha);
```

Description Each of the `mllib_GraphicsFillEllipse_*`() functions draws a filled ellipse with the center at (x, y) , major semiaxis a , and minor semiaxis b . The angle of the major semiaxis is t counterclockwise from the X axis.

Each of the `mllib_GraphicsFillEllipse_X_*`() functions draws a filled ellipse in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsFillEllipse_A_*`() functions draws a filled ellipse with antialiasing.

Each of the `mllib_GraphicsFillEllipse_B_*`() functions draws a filled ellipse with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - \alpha) + c * \alpha) / 255$$

Each of the `mllib_GraphicsFillEllipse_A_*`() functions draws a filled ellipse with antialiasing and alpha blending.

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>x</i>	X coordinate of the center.
<i>y</i>	Y coordinate of the center.
<i>a</i>	Major semiaxis of the ellipse.
<i>b</i>	Minor semiaxis of the ellipse.
<i>t</i>	Angle of major semiaxis in radians.
<i>c</i>	Color used in the drawing.
<i>c2</i>	Alternation color.
<i>alpha</i>	Alpha value for blending. $0 \leq \alpha \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillArc\(3MLIB\)](#), [mllib_GraphicsFillCircle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillPolygon, mllib_GraphicsFillPolygon_8, mllib_GraphicsFillPolygon_32, mllib_GraphicsFillPolygon_X_8, mllib_GraphicsFillPolygon_X_32, mllib_GraphicsFillPolygon_A_8, mllib_GraphicsFillPolygon_A_32, mllib_GraphicsFillPolygon_B_8, mllib_GraphicsFillPolygon_B_32, mllib_GraphicsFillPolygon_G_8, mllib_GraphicsFillPolygon_G_32, mllib_GraphicsFillPolygon_Z_8, mllib_GraphicsFillPolygon_Z_32, mllib_GraphicsFillPolygon_AB_8, mllib_GraphicsFillPolygon_AB_32, mllib_GraphicsFillPolygon_ABG_8, mllib_GraphicsFillPolygon_ABG_32, mllib_GraphicsFillPolygon_ABGZ_8, mllib_GraphicsFillPolygon_ABGZ_32, mllib_GraphicsFillPolygon_ABZ_8, mllib_GraphicsFillPolygon_ABZ_32, mllib_GraphicsFillPolygon_AG_8, mllib_GraphicsFillPolygon_AG_32, mllib_GraphicsFillPolygon_AGZ_8, mllib_GraphicsFillPolygon_AGZ_32, mllib_GraphicsFillPolygon_AZ_8, mllib_GraphicsFillPolygon_AZ_32, mllib_GraphicsFillPolygon_BG_8, mllib_GraphicsFillPolygon_BG_32, mllib_GraphicsFillPolygon_BGZ_8, mllib_GraphicsFillPolygon_BGZ_32, mllib_GraphicsFillPolygon_BZ_8, mllib_GraphicsFillPolygon_BZ_32, mllib_GraphicsFillPolygon_GZ_8, mllib_GraphicsFillPolygon_GZ_32 – draw filled polygon

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_GraphicsFillPolygon_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_X_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mllib_status mllib_GraphicsFillPolygon_X_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mllib_status mllib_GraphicsFillPolygon_A_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_A_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_B_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_B_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
```

```

mllib_status mllib_GraphicsFillPolygon_G_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_G_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_Z_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_Z_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_AB_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_AB_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABG_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
      const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABGZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABGZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABZ_8(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints,
      mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_ABZ_32(mllib_image *buffer,
      mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
      const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_AG_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y,
      mlib_s32 npoints, const mlib_s32 *c);

```

```
mllib_status mllib_GraphicsFillPolygon_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillPolygon_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillPolygon_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillPolygon_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
```

```
const mlib_s16 *z, mlib_s32 npoints,
const mlib_s32 *c);
```

Description Each of the `mllib_GraphicsFillPolygon_*`() functions draws a filled polygon enclosing (x1,y1), (x2,y2), ..., and (xn,yn).

Each of the `mllib_GraphicsFillPolygon_X_*`() functions draws a filled polygon in Xor mode as follows:

$$\text{data}[x,y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsFillPolygon_A_*`() functions draws a filled polygon with antialiasing.

Each of the `mllib_GraphicsFillPolygon_B_*`() functions draws a filled polygon with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillPolygon_G_*`() functions draws a filled polygon with Gouraud shading.

Each of the `mllib_GraphicsFillPolygon_Z_*`() functions draws a filled polygon with Z buffering.

Each of the other functions draws a filled polygon with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x Pointer to the array of X coordinates of the vertices.

y Pointer to the array of Y coordinates of the vertices.

z Pointer to the array of Z coordinates of the vertices.

npoints Number of vertices in the arrays.

c Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawPolygon\(3MLIB\)](#), [attributes\(5\)](#)

REFERENCE

Multimedia Library Functions - Part 2

Name mllib_GraphicsFillRectangle, mllib_GraphicsFillRectangle_8, mllib_GraphicsFillRectangle_32, mllib_GraphicsFillRectangle_X_8, mllib_GraphicsFillRectangle_X_32, mllib_GraphicsFillRectangle_B_8, mllib_GraphicsFillRectangle_B_32 – draw filled rectangle

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsFillRectangle_8(mllib_image *buffer, mlib_s16 x,
                                           mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);

mllib_status mllib_GraphicsFillRectangle_32(mllib_image *buffer, mlib_s16 x,
                                           mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);

mllib_status mllib_GraphicsFillRectangle_X_8(mllib_image *buffer, mlib_s16 x,
                                           mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 c2);

mllib_status mllib_GraphicsFillRectangle_X_32(mllib_image *buffer,
                                           mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c,
                                           mlib_s32 c2);

mllib_status mllib_GraphicsFillRectangle_B_8(mllib_image *buffer, mlib_s16 x,
                                           mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillRectangle_B_32(mllib_image *buffer,
                                           mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c,
                                           mlib_s32 a);
```

Description Each of the `mllib_GraphicsFillRectangle_*`() functions draws a filled rectangle with the upper-left corner at (x, y) , width w , and height h .

Each of the `mllib_GraphicsFillRectangle_X_*`() functions draws a filled rectangle in Xor mode as follows:

$$\text{data}[x, y] \hat{=} c \wedge c2$$

Each of the `mllib_GraphicsFillRectangle_B_*`() functions draws a filled rectangle with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

x X coordinate of the upper-left corner of the rectangle.

y Y coordinate of the upper-left corner of the rectangle.

w Width of the rectangle.

h Height of the rectangle.

c Color used in the drawing.

c2 Alternation color.

a Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_GraphicsDrawRectangle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillTriangle, mllib_GraphicsFillTriangle_8, mllib_GraphicsFillTriangle_32, mllib_GraphicsFillTriangle_X_8, mllib_GraphicsFillTriangle_X_32, mllib_GraphicsFillTriangle_A_8, mllib_GraphicsFillTriangle_A_32, mllib_GraphicsFillTriangle_B_8, mllib_GraphicsFillTriangle_B_32, mllib_GraphicsFillTriangle_G_8, mllib_GraphicsFillTriangle_G_32, mllib_GraphicsFillTriangle_Z_8, mllib_GraphicsFillTriangle_Z_32, mllib_GraphicsFillTriangle_AB_8, mllib_GraphicsFillTriangle_AB_32, mllib_GraphicsFillTriangle_ABG_8, mllib_GraphicsFillTriangle_ABG_32, mllib_GraphicsFillTriangle_ABGZ_8, mllib_GraphicsFillTriangle_ABGZ_32, mllib_GraphicsFillTriangle_ABZ_8, mllib_GraphicsFillTriangle_ABZ_32, mllib_GraphicsFillTriangle_AG_8, mllib_GraphicsFillTriangle_AG_32, mllib_GraphicsFillTriangle_AGZ_8, mllib_GraphicsFillTriangle_AGZ_32, mllib_GraphicsFillTriangle_AZ_8, mllib_GraphicsFillTriangle_AZ_32, mllib_GraphicsFillTriangle_BG_8, mllib_GraphicsFillTriangle_BG_32, mllib_GraphicsFillTriangle_BGZ_8, mllib_GraphicsFillTriangle_BGZ_32, mllib_GraphicsFillTriangle_BZ_8, mllib_GraphicsFillTriangle_BZ_32, mllib_GraphicsFillTriangle_GZ_8, mllib_GraphicsFillTriangle_GZ_32 – draw filled triangle

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_GraphicsFillTriangle_8(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c);

mllib_status mllib_GraphicsFillTriangle_32(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c);

mllib_status mllib_GraphicsFillTriangle_X_8(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c, mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangle_X_32(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c, mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangle_A_8(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c);

mllib_status mllib_GraphicsFillTriangle_A_32(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangle_B_8(mllib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c,
    mlib_s32 a);
```

```

mlib_status mlib_GraphicsFillTriangle_B_32(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c,
    mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_G_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_G_32(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_AB_8(mlib_image *buffer, mlib_s16 x1,
    mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_AB_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABG_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
    mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c,
    mlib_s32 a);

```

```
mllib_status mllib_GraphicsFillTriangle_ABZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c, mllib_s32 a);

mllib_status mllib_GraphicsFillTriangle_AG_8(mllib_image *buffer, mllib_s16 x1,
        mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
        mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsFillTriangle_AG_32(mllib_image *buffer,
        mllib_s16 x1, mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3,
        mllib_s16 y3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsFillTriangle_AGZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsFillTriangle_AGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3);

mllib_status mllib_GraphicsFillTriangle_AZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c);

mllib_status mllib_GraphicsFillTriangle_AZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c);

mllib_status mllib_GraphicsFillTriangle_BG_8(mllib_image *buffer, mllib_s16 x1,
        mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3, mllib_s16 y3,
        mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsFillTriangle_BG_32(mllib_image *buffer,
        mllib_s16 x1, mllib_s16 y1, mllib_s16 x2, mllib_s16 y2, mllib_s16 x3,
        mllib_s16 y3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsFillTriangle_BGZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsFillTriangle_BGZ_32(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
        mllib_s16 x2, mllib_s16 y2, mllib_s16 z2, mllib_s16 x3, mllib_s16 y3,
        mllib_s16 z3, mllib_s32 c1, mllib_s32 c2, mllib_s32 c3, mllib_s32 a);

mllib_status mllib_GraphicsFillTriangle_BZ_8(mllib_image *buffer,
        mllib_image *zbuffer, mllib_s16 x1, mllib_s16 y1, mllib_s16 z1,
```

```

        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangle_BZ_32(mlib_image *buffer,
        mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangle_GZ_8(mlib_image *buffer,
        mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
        mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mllib_status mllib_GraphicsFillTriangle_GZ_32(mlib_image *buffer,
        mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
        mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
        mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

```

Description Each of the `mllib_GraphicsFillTriangle_*()` functions draws a filled triangle with the vertices at (x1,y1), (x2,y2), and (x3,y3).

Each of the `mllib_GraphicsFillTriangle_X_*()` functions draws a filled triangle in Xor mode as follows:

$$\text{data}[x,y] \wedge c \wedge c2$$

Each of the `mllib_GraphicsFillTriangle_A_*()` functions draws a filled triangle with antialiasing.

Each of the `mllib_GraphicsFillTriangle_B_*()` functions draws a filled triangle with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillTriangle_G_*()` functions draws a filled triangle with Gouraud shading.

Each of the `mllib_GraphicsFillTriangle_Z_*()` functions draws a filled triangle with Z buffering.

Each of the other functions draws a filled triangle with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x1 X coordinate of the first vertex.

y1 Y coordinate of the first vertex.

- z1* Z coordinate of the first vertex.
- x2* X coordinate of the second vertex.
- y2* Y coordinate of the second vertex.
- z2* Z coordinate of the second vertex.
- x3* X coordinate of the third vertex.
- y3* Y coordinate of the third vertex.
- z3* Z coordinate of the third vertex.
- c* Color used in the drawing.
- c1* Color of the first vertex.
- c2* Color of the second vertex, or the alternation color in Xor Mode.
- c3* Color of the third vertex.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsDrawTriangle\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillTriangleFanSet, mllib_GraphicsFillTriangleFanSet_8, mllib_GraphicsFillTriangleFanSet_32, mllib_GraphicsFillTriangleFanSet_X_8, mllib_GraphicsFillTriangleFanSet_X_32, mllib_GraphicsFillTriangleFanSet_A_8, mllib_GraphicsFillTriangleFanSet_A_32, mllib_GraphicsFillTriangleFanSet_B_8, mllib_GraphicsFillTriangleFanSet_B_32, mllib_GraphicsFillTriangleFanSet_G_8, mllib_GraphicsFillTriangleFanSet_G_32, mllib_GraphicsFillTriangleFanSet_Z_8, mllib_GraphicsFillTriangleFanSet_Z_32, mllib_GraphicsFillTriangleFanSet_AB_8, mllib_GraphicsFillTriangleFanSet_AB_32, mllib_GraphicsFillTriangleFanSet_ABG_8, mllib_GraphicsFillTriangleFanSet_ABG_32, mllib_GraphicsFillTriangleFanSet_ABGZ_8, mllib_GraphicsFillTriangleFanSet_ABGZ_32, mllib_GraphicsFillTriangleFanSet_ABZ_8, mllib_GraphicsFillTriangleFanSet_ABZ_32, mllib_GraphicsFillTriangleFanSet_AG_8, mllib_GraphicsFillTriangleFanSet_AG_32, mllib_GraphicsFillTriangleFanSet_AGZ_8, mllib_GraphicsFillTriangleFanSet_AGZ_32, mllib_GraphicsFillTriangleFanSet_AZ_8, mllib_GraphicsFillTriangleFanSet_AZ_32, mllib_GraphicsFillTriangleFanSet_BG_8, mllib_GraphicsFillTriangleFanSet_BG_32, mllib_GraphicsFillTriangleFanSet_BGZ_8, mllib_GraphicsFillTriangleFanSet_BGZ_32, mllib_GraphicsFillTriangleFanSet_BZ_8, mllib_GraphicsFillTriangleFanSet_BZ_32, mllib_GraphicsFillTriangleFanSet_GZ_8, mllib_GraphicsFillTriangleFanSet_GZ_32 – draw filled triangle set where all members of the set have a common vertex

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```

mllib_status mllib_GraphicsFillTriangleFanSet_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_X_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleFanSet_X_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleFanSet_A_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_A_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_B_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_B_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

```

```
mllib_status mllib_GraphicsFillTriangleFanSet_G_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleFanSet_G_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleFanSet_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleFanSet_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleFanSet_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleFanSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
```



```

mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleFanSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleFanSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mlib_GraphicsFillTriangleFanSet_X_*` functions draws a set of filled triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, $\{(x_1, y_1), (x_3, y_3), (x_4, y_4)\}$, ..., and $\{(x_1, y_1), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mlib_GraphicsFillTriangleFanSet_X_*` functions draws a set of filled triangles in Xor mode as follows:

$$\text{data}[x,y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsFillTriangleFanSet_A_*`() functions draws a set of filled triangles with antialiasing.

Each of the `mllib_GraphicsFillTriangleFanSet_B_*`() functions draws a set of filled triangles with alpha blending as follows:

$$\text{data}[x,y] = (\text{data}[x,y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillTriangleFanSet_G_*`() functions draws a set of filled triangles with Gouraud shading.

Each of the `mllib_GraphicsFillTriangleFanSet_Z_*`() functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x* Pointer to array of X coordinates of the points.
- y* Pointer to array of Y coordinates of the points.
- z* Pointer to array of Z coordinates of the points.
- npoints* Number of points in the arrays.
- c* Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_GraphicsFillTriangle(3MLIB)`, `mlib_GraphicsFillTriangleSet(3MLIB)`,
`mlib_GraphicsFillTriangleStripSet(3MLIB)`, `attributes(5)`

Name mllib_GraphicsFillTriangleSet, mllib_GraphicsFillTriangleSet_8, mllib_GraphicsFillTriangleSet_32, mllib_GraphicsFillTriangleSet_X_8, mllib_GraphicsFillTriangleSet_X_32, mllib_GraphicsFillTriangleSet_A_8, mllib_GraphicsFillTriangleSet_A_32, mllib_GraphicsFillTriangleSet_B_8, mllib_GraphicsFillTriangleSet_B_32, mllib_GraphicsFillTriangleSet_G_8, mllib_GraphicsFillTriangleSet_G_32, mllib_GraphicsFillTriangleSet_Z_8, mllib_GraphicsFillTriangleSet_Z_32, mllib_GraphicsFillTriangleSet_AB_8, mllib_GraphicsFillTriangleSet_AB_32, mllib_GraphicsFillTriangleSet_ABG_8, mllib_GraphicsFillTriangleSet_ABG_32, mllib_GraphicsFillTriangleSet_ABGZ_8, mllib_GraphicsFillTriangleSet_ABGZ_32, mllib_GraphicsFillTriangleSet_ABZ_8, mllib_GraphicsFillTriangleSet_ABZ_32, mllib_GraphicsFillTriangleSet_AG_8, mllib_GraphicsFillTriangleSet_AG_32, mllib_GraphicsFillTriangleSet_AGZ_8, mllib_GraphicsFillTriangleSet_AGZ_32, mllib_GraphicsFillTriangleSet_AZ_8, mllib_GraphicsFillTriangleSet_AZ_32, mllib_GraphicsFillTriangleSet_BG_8, mllib_GraphicsFillTriangleSet_BG_32, mllib_GraphicsFillTriangleSet_BGZ_8, mllib_GraphicsFillTriangleSet_BGZ_32, mllib_GraphicsFillTriangleSet_BZ_8, mllib_GraphicsFillTriangleSet_BZ_32, mllib_GraphicsFillTriangleSet_GZ_8, mllib_GraphicsFillTriangleSet_GZ_32 – draw filled triangle set where each member can have different vertices

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsFillTriangleSet_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_X_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleSet_X_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleSet_A_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_A_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_B_8(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_B_32(mllib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);
```

```

mllib_status mllib_GraphicsFillTriangleSet_G_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_G_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_Z_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_Z_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_AB_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_AB_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
        mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABG_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABGZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABGZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABZ_8(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_ABZ_32(mlib_image *buffer,
        mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
        const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_AG_8(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_AG_32(mlib_image *buffer,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
        const mlib_s32 *c);

```

```

mllib_status mllib_GraphicsFillTriangleSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_BZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsFillTriangleSet_*`() functions draws a set of filled triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}, \{(x_4, y_4), (x_5, y_5), (x_6, y_6)\}, \dots$, and $\{(x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mllib_GraphicsFillTriangleSet_X_*`() functions draws a set of filled triangles in Xor mode as follows:

```
data[x,y] ^= c ^ c2
```

Each of the `mllib_GraphicsFillTriangleSet_A_*`() functions draws a set of filled triangles with antialiasing.

Each of the `mllib_GraphicsFillTriangleSet_B_*`() functions draws a set of filled triangles with alpha blending as follows:

```
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
```

Each of the `mllib_GraphicsFillTriangleSet_G_*`() functions draws a set of filled triangles with Gouraud shading.

Each of the `mllib_GraphicsFillTriangleSet_Z_*`() functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- zbuffer* Pointer to the image that holds the Z buffer.
- x* Pointer to array of X coordinates of the points.
- y* Pointer to array of Y coordinates of the points.
- z* Pointer to array of Z coordinates of the points.
- npoints* Number of points in the arrays. *npoints* must be a multiple of 3.
- c* Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- c2* Alternation color.
- a* Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_GraphicsFillTriangle\(3MLIB\)](#), [mllib_GraphicsFillTriangleFanSet\(3MLIB\)](#),
[mllib_GraphicsFillTriangleStripSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFillTriangleStripSet, mllib_GraphicsFillTriangleStripSet_8, mllib_GraphicsFillTriangleStripSet_32, mllib_GraphicsFillTriangleStripSet_X_8, mllib_GraphicsFillTriangleStripSet_X_32, mllib_GraphicsFillTriangleStripSet_A_8, mllib_GraphicsFillTriangleStripSet_A_32, mllib_GraphicsFillTriangleStripSet_B_8, mllib_GraphicsFillTriangleStripSet_B_32, mllib_GraphicsFillTriangleStripSet_G_8, mllib_GraphicsFillTriangleStripSet_G_32, mllib_GraphicsFillTriangleStripSet_Z_8, mllib_GraphicsFillTriangleStripSet_Z_32, mllib_GraphicsFillTriangleStripSet_AB_8, mllib_GraphicsFillTriangleStripSet_AB_32, mllib_GraphicsFillTriangleStripSet_ABG_8, mllib_GraphicsFillTriangleStripSet_ABG_32, mllib_GraphicsFillTriangleStripSet_ABGZ_8, mllib_GraphicsFillTriangleStripSet_ABGZ_32, mllib_GraphicsFillTriangleStripSet_ABZ_8, mllib_GraphicsFillTriangleStripSet_ABZ_32, mllib_GraphicsFillTriangleStripSet_AG_8, mllib_GraphicsFillTriangleStripSet_AG_32, mllib_GraphicsFillTriangleStripSet_AGZ_8, mllib_GraphicsFillTriangleStripSet_AGZ_32, mllib_GraphicsFillTriangleStripSet_AZ_8, mllib_GraphicsFillTriangleStripSet_AZ_32, mllib_GraphicsFillTriangleStripSet_BG_8, mllib_GraphicsFillTriangleStripSet_BG_32, mllib_GraphicsFillTriangleStripSet_BGZ_8, mllib_GraphicsFillTriangleStripSet_BGZ_32, mllib_GraphicsFillTriangleStripSet_BZ_8, mllib_GraphicsFillTriangleStripSet_BZ_32, mllib_GraphicsFillTriangleStripSet_GZ_8, mllib_GraphicsFillTriangleStripSet_GZ_32 – draw filled triangle set where the first side of each member is common to the second side of the previous member

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_GraphicsFillTriangleStripSet_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_X_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleStripSet_X_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mllib_status mllib_GraphicsFillTriangleStripSet_A_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_A_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_B_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_B_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
```

```
mllib_status mllib_GraphicsFillTriangleStripSet_G_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_G_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c,
    mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
```

```

mllib_status mllib_GraphicsFillTriangleStripSet_AG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_AGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_AGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_AZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_AZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mllib_status mllib_GraphicsFillTriangleStripSet_BG_8(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_BG_32(mllib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_BGZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_BGZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_BZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_BZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mllib_status mllib_GraphicsFillTriangleStripSet_GZ_8(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mllib_status mllib_GraphicsFillTriangleStripSet_GZ_32(mllib_image *buffer,
    mllib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

```

Description Each of the `mllib_GraphicsFillTriangleStripSet_*()` functions draws a set of filled triangles with vertices at $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, $\{(x_2, y_2), (x_3, y_3), (x_4, y_4)\}$, ..., and $\{(x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\}$.

Each of the `mllib_GraphicsFillTriangleStripSet_X_*()` functions draws a set of filled triangles in Xor mode as follows:

$$\text{data}[x, y] \wedge= c \wedge c2$$

Each of the `mllib_GraphicsFillTriangleStripSet_A_*()` functions draws a set of filled triangles with antialiasing.

Each of the `mllib_GraphicsFillTriangleStripSet_B_*()` functions draws a set of filled triangles with alpha blending as follows:

$$\text{data}[x, y] = (\text{data}[x, y] * (255 - a) + c * a) / 255$$

Each of the `mllib_GraphicsFillTriangleStripSet_G_*()` functions draws a set of filled triangles with Gouraud shading.

Each of the `mllib_GraphicsFillTriangleStripSet_Z_*()` functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

<i>buffer</i>	Pointer to the image into which the function is drawing.
<i>zbuffer</i>	Pointer to the image that holds the Z buffer.
<i>x</i>	Pointer to array of X coordinates of the points.
<i>y</i>	Pointer to array of Y coordinates of the points.
<i>z</i>	Pointer to array of Z coordinates of the points.
<i>npoints</i>	Number of points in the arrays.
<i>c</i>	Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
<i>c2</i>	Alternation color.
<i>a</i>	Alpha value for blending. $0 \leq a \leq 255$.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_GraphicsFillTriangle\(3MLIB\)](#), [mlib_GraphicsFillTriangleSet\(3MLIB\)](#),
[mlib_GraphicsFillTriangleFanSet\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_GraphicsFloodFill_8, mllib_GraphicsFloodFill_32 – flood fill

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_GraphicsFloodFill_8(mllib_image *buffer, mllib_s16 x,
    mllib_s16 y, mllib_s32 c, mllib_s32 c2);

mllib_status mllib_GraphicsFloodFill_32(mllib_image *buffer, mllib_s16 x,
    mllib_s16 y, mllib_s32 c, mllib_s32 c2);
```

Description Each of these functions performs flood fill.

Parameters Each of the functions takes the following arguments:

- buffer* Pointer to the image into which the function is drawing.
- x* X coordinate of the starting point.
- y* Y coordinate of the starting point.
- c* Color used in the drawing.
- c2* Color that defines the filling interior.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [attributes\(5\)](#)

Name mllib_ImageAbs – computes the absolute value of the image pixels

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageAbs(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageAbs()` function computes the absolute value of the image pixels.

It uses the following equation:

$$dst[x][y][i] = |src[x][y][i]|$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAbs_Fp\(3MLIB\)](#), [mllib_ImageAbs_Fp_Inp\(3MLIB\)](#), [mllib_ImageAbs_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAbs_Fp – computes the absolute value of the image pixels

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageAbs_Fp(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageAbs_Fp()` function computes the floating-point absolute value of the image pixels.

It uses the following equation:

`dst[x][y][i] = |src[x][y][i]|`

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAbs\(3MLIB\)](#), [mllib_ImageAbs_Fp_Inp\(3MLIB\)](#), [mllib_ImageAbs_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAbs_Fp_Inp – computes the absolute value of the image pixels

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAbs_Fp_Inp(mllib_image *srcdst);
```

Description The `mllib_ImageAbs_Fp_Inp()` function computes the floating-point absolute value of the image pixels, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = |\text{srcdst}[x][y][i]|$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAbs\(3MLIB\)](#), [mllib_ImageAbs_Fp\(3MLIB\)](#), [mllib_ImageAbs_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAbs_Inp – computes the absolute value of the image pixels, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageAbs_Inp(mllib_image *srcdst);
```

Description The `mllib_ImageAbs_Inp()` function computes the absolute value of the image pixels in place. It uses the following equation:

$$srcdst[x][y][i] = |srcdst[x][y][i]|$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAbs\(3MLIB\)](#), [mllib_ImageAbs_Fp\(3MLIB\)](#), [mllib_ImageAbs_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAdd – computes the addition of two images on a pixel-by-pixel basis

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAdd(mllib_image dst, const mllib_image *src1,
                           const mllib_image *src2);
```

Description The `mllib_ImageAdd()` function computes the addition of two images on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] + src2[x][y][i]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src1 Pointer to first source image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAdd_Fp\(3MLIB\)](#), [mllib_ImageAdd_Fp_Inp\(3MLIB\)](#), [mllib_ImageAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAdd_Fp – computes the addition of two images on a pixel-by-pixel basis

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAdd_Fp(mllib_image dst, const mllib_image *src1,
    const mllib_image *src2);
```

Description The `mllib_ImageAdd_Fp()` function computes the addition of two floating-point images on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] + src2[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to first source image.
- src2* Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAdd\(3MLIB\)](#), [mllib_ImageAdd_Fp_Inp\(3MLIB\)](#), [mllib_ImageAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAdd_Fp_Inp – computes the addition of two images on a pixel-by-pixel basis

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAdd_Fp_Inp(mllib_image *src1dst,
    const mllib_image *src2);
```

Description The `mllib_ImageAdd_Fp_Inp()` function computes the addition of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] + \text{src2}[x][y][i]$$

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTEVALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAdd\(3MLIB\)](#), [mllib_ImageAdd_Fp\(3MLIB\)](#), [mllib_ImageAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAdd_Inp – computes the addition of two images on a pixel-by-pixel basis, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAdd_Inp(mllib_image *src1dst, const mllib_image *src2);
```

Description The `mllib_ImageAdd_Inp()` function computes the addition of two images on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src1dst[x][y][i] = src1dst[x][y][i] + src2[x][y][i]
```

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAdd\(3MLIB\)](#), [mllib_ImageAdd_Fp\(3MLIB\)](#), [mllib_ImageAdd_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageAffine – image affine transformation

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```

```
mlib_status mlib_ImageAffine(mlib_image *dst, const mlib_image *src,
    const mlib_d64 *mtx, mlib_filter filter, mlib_edge edge);
```

Description The `mlib_ImageAffine()` function does affine transformation on an image according to the following equation:

$$\begin{aligned} x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y \end{aligned}$$

where a point with coordinates (x_s, y_s) in the source image is mapped to a point with coordinates (x_d, y_d) in the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds a ; `mtx[1]` holds b ; `mtx[2]` holds t_x ; `mtx[3]` holds c ; `mtx[4]` holds d ; `mtx[5]` holds t_y .

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine_Fp\(3MLIB\)](#), [mllib_ImageAffineIndex\(3MLIB\)](#),
[mllib_ImageAffineTransform\(3MLIB\)](#), [mllib_ImageAffineTransform_Fp\(3MLIB\)](#),
[mllib_ImageAffineTransformIndex\(3MLIB\)](#), [mllib_ImageSetPaddings\(3MLIB\)](#),
[attributes\(5\)](#)

Name mlib_ImageAffine_Fp – image affine transformation

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageAffine_Fp(mlib_image *dst, const mlib_image *src,
                                const mlib_d64 *mtx, mlib_filter filter, mlib_edge edge);
```

Description The `mlib_ImageAffine_Fp()` function does affine transformation on a floating-point image according to the following equation:

$$\begin{aligned} x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y \end{aligned}$$

where a point with coordinates (x_s , y_s) in the source image is mapped to a point with coordinates (x_d , y_d) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds a ; `mtx[1]` holds b ; `mtx[2]` holds t_x ; `mtx[3]` holds c ; `mtx[4]` holds d ; `mtx[5]` holds t_y .

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine\(3MLIB\)](#), [mllib_ImageAffineIndex\(3MLIB\)](#),
[mllib_ImageAffineTransform\(3MLIB\)](#), [mllib_ImageAffineTransform_Fp\(3MLIB\)](#),
[mllib_ImageAffineTransformIndex\(3MLIB\)](#), [mllib_ImageSetPaddings\(3MLIB\)](#),
[attributes\(5\)](#)

Name mllib_ImageAffineTransform – affine transformation on a color indexed image

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageAffineTransform(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *mtx, mllib_filter filter, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageAffineTransform()` function does affine transformation on a color indexed image according to the following equation:

$$\begin{aligned} x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y \end{aligned}$$

where a point with coordinates (x_s, y_s) in the source image is mapped to a point with coordinates (x_d, y_d) in the destination image.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds *a*; `mtx[1]` holds *b*; `mtx[2]` holds *t_x*; `mtx[3]` holds *c*; `mtx[4]` holds *d*; `mtx[5]` holds *t_y*.

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine\(3MLIB\)](#), [mllib_ImageAffine_Fp\(3MLIB\)](#),
[mllib_ImageAffineTransform\(3MLIB\)](#), [mllib_ImageAffineTransform_Fp\(3MLIB\)](#),
[mllib_ImageAffineTransformIndex\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAffineTable – affine transformation on an image with table-driven interpolation

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAffineTable(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *mtx, const void *interp_table, mllib_edge edge);
```

Description The `mllib_ImageAffineTable()` function does affine transformation on an image with table-driven interpolation.

The following equation represents the affine transformation:

$$\begin{aligned} x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y \end{aligned}$$

where a point with coordinates (x_s, y_s) in the source image is mapped to a point with coordinates (x_d, y_d) in the destination image.

The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>mtx</i>	Transformation matrix. <code>mtx[0]</code> holds <i>a</i> ; <code>mtx[1]</code> holds <i>b</i> ; <code>mtx[2]</code> holds <i>t_x</i> ; <code>mtx[3]</code> holds <i>c</i> ; <code>mtx[4]</code> holds <i>d</i> ; <code>mtx[5]</code> holds <i>t_y</i> .
<i>interp_table</i>	Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.
<i>edge</i>	Type of edge condition. It can be one of the following: <code>MLIB_EDGE_DST_NO_WRITE</code> <code>MLIB_EDGE_DST_FILL_ZERO</code> <code>MLIB_EDGE_OP_NEAREST</code> <code>MLIB_EDGE_SRC_EXTEND</code> <code>MLIB_EDGE_SRC_PADDED</code>

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInterpTableCreate\(3MLIB\)](#), [mllib_ImageInterpTableDelete\(3MLIB\)](#),
[mllib_ImageAffineTable_Fp\(3MLIB\)](#), [mllib_ImageAffine\(3MLIB\)](#),
[mllib_ImageAffine_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAffineTransform_Fp – affine transformation on an image with table-driven interpolation

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageAffineTransform_Fp(mllib_image *dst,  
      const mllib_image *src, const mllib_d64 *mtx,  
      const void *interp_table, mllib_edge edge);
```

Description The `mllib_ImageAffineTransform_Fp()` function does affine transformation on a floating-point image with table-driven interpolation.

The following equation represents the affine transformation:

$$\begin{aligned}x_d &= a*x_s + b*y_s + t_x \\y_d &= c*x_s + d*y_s + t_y\end{aligned}$$

where a point with coordinates (x_s, y_s) in the source image is mapped to a point with coordinates (x_d, y_d) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>mtx</i>	Transformation matrix. <code>mtx[0]</code> holds <i>a</i> ; <code>mtx[1]</code> holds <i>b</i> ; <code>mtx[2]</code> holds <i>t_x</i> ; <code>mtx[3]</code> holds <i>c</i> ; <code>mtx[4]</code> holds <i>d</i> ; <code>mtx[5]</code> holds <i>t_y</i> .
<i>interp_table</i>	Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.
<i>edge</i>	Type of edge condition. It can be one of the following: <code>MLIB_EDGE_DST_NO_WRITE</code> <code>MLIB_EDGE_DST_FILL_ZERO</code> <code>MLIB_EDGE_OP_NEAREST</code> <code>MLIB_EDGE_SRC_EXTEND</code> <code>MLIB_EDGE_SRC_PADDED</code>

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInterpTableCreate\(3MLIB\)](#), [mllib_ImageInterpTableDelete\(3MLIB\)](#),
[mllib_ImageAffineTable\(3MLIB\)](#), [mllib_ImageAffine\(3MLIB\)](#),
[mllib_ImageAffine_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAffineTransform – affine transformation on an image, checking the matrix first

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageAffineTransform(mllib_image *dst,
    const mllib_image *src, const mllib_d64 *mtx, mllib_filter filter,
    mllib_edge edge);
```

Description The `mllib_ImageAffineTransform()` function does affine transformation on an image, checking the matrix first and taking advantage of special cases.

The following equation represents the affine transformation:

$$\begin{aligned}x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y\end{aligned}$$

where a point with coordinates (x_s, y_s) in the source image is mapped to a point with coordinates (x_d, y_d) in the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds *a*; `mtx[1]` holds *b*; `mtx[2]` holds *t_x*; `mtx[3]` holds *c*; `mtx[4]` holds *d*; `mtx[5]` holds *t_y*.

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine\(3MLIB\)](#), [mllib_ImageAffine_Fp\(3MLIB\)](#),
[mllib_ImageAffineIndex\(3MLIB\)](#), [mllib_ImageAffineTransform_Fp\(3MLIB\)](#),
[mllib_ImageAffineTransformIndex\(3MLIB\)](#), [mllib_ImageSetPaddings\(3MLIB\)](#),
[attributes\(5\)](#)

Name mllib_ImageAffineTransform_Fp – affine transformation on an image, checking the matrix first

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageAffineTransform_Fp(mllib_image *dst,
      const mllib_image *src, const mllib_d64 *mtx, mllib_filter filter,
      mllib_edge edge);
```

Description The `mllib_ImageAffineTransform_Fp()` function does affine transformation on a floating-point image, checking the matrix first and taking advantage of special cases.

The following equation represents the affine transformation:

$$\begin{aligned}x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y\end{aligned}$$

where a point with coordinates (x_s , y_s) in the source image is mapped to a point with coordinates (x_d , y_d) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds a ; `mtx[1]` holds b ; `mtx[2]` holds t_x ; `mtx[3]` holds c ; `mtx[4]` holds d ; `mtx[5]` holds t_y .

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine\(3MLIB\)](#), [mllib_ImageAffine_Fp\(3MLIB\)](#),
[mllib_ImageAffineIndex\(3MLIB\)](#), [mllib_ImageAffineTransform\(3MLIB\)](#),
[mllib_ImageAffineTransformIndex\(3MLIB\)](#), [mllib_ImageSetPaddings\(3MLIB\)](#),
[attributes\(5\)](#)

Name mllib_ImageAffineTransformIndex – affine transformation on a color indexed image, checking the matrix first

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageAffineTransformIndex(mllib_image *dst,
      const mllib_image *src, const mllib_d64 *mtx, mllib_filter filter,
      mllib_edge edge, const void *colormap);
```

Description The `mllib_ImageAffineTransformIndex()` function does affine transformation on a color indexed image, checking the matrix first and taking advantage of special cases.

The following equation represents the affine transformation:

$$\begin{aligned} x_d &= a*x_s + b*y_s + t_x \\ y_d &= c*x_s + d*y_s + t_y \end{aligned}$$

where a point with coordinates (x_s , y_s) in the source image is mapped to a point with coordinates (x_d , y_d) in the destination image.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mtx Transformation matrix. `mtx[0]` holds a ; `mtx[1]` holds b ; `mtx[2]` holds t_x ; `mtx[3]` holds c ; `mtx[4]` holds d ; `mtx[5]` holds t_y .

filter Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAffine\(3MLIB\)](#), [mllib_ImageAffine_Fp\(3MLIB\)](#),
[mllib_ImageAffineIndex\(3MLIB\)](#), [mllib_ImageAffineTransform\(3MLIB\)](#),
[mllib_ImageAffineTransform_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageAnd – computes the And of two images

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageAnd(mlib_image *dst, const mlib_image *src1,
                          const mlib_image *src2);
```

Description The `mlib_ImageAnd()` function computes the And of two images according to the following equation:

$$\text{dst}[x][y][i] = \text{src1}[x][y][i] \ \& \ \text{src2}[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.
src1 Pointer to first source image.
src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageAnd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAnd_Inp – computes the And of two image, in place

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageAnd_Inp(mllib_image *src1dst, const mllib_image *src2);`

Description The `mllib_ImageAnd_Inp()` function computes the And of two images, in place, according to the following equation:

`src1dst[x][y][i] = src1dst[x][y][i] & src2[x][y][i]`

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAnd\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAndNot1_Inp – computes the And of the first source image and the Not of the second source image, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAndNot1_Inp(mllib_image *src1dst,
    const mllib_image *src2);
```

Description The `mllib_ImageAndNot1_Inp()` function computes the logical Not of the second source image and then computes the logical And of that result with the first source image, on a pixel-by-pixel basis, and stores the final result in the first source image. It uses the following equation:

$$src1dst[x][y][i] = src1dst[x][y][i] \& (\sim src2[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAndNot\(3MLIB\)](#), [mllib_ImageAndNot2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAndNot2_Inp – computes the And of the first source image and the Not of the second source image, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAndNot2_Inp(mllib_image *src2dst, const mllib_image *
    src1);
```

Description The mllib_ImageAndNot2_Inp() function computes the logical Not of the second source image and then computes the logical And of that result with the first source image, on a pixel-by-pixel basis, and stores the final result in the second source image. It uses the following equation:

$$src2dst[x][y][i] = src1[x][y][i] \ \& \ (\sim src2dst[x][y][i])$$

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

Parameters The function takes the following arguments:

src2dst Pointer to second source and destination image.

src1 Pointer to first source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAnd\(3MLIB\)](#), [mllib_ImageAnd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAndNot – computes the And of the first source image and the Not of the second source image

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageAndNot(mllib_image *dst, const mllib_image *src1,  
                               const mllib_image *src2);
```

Description The `mllib_ImageAndNot()` function computes the logical Not of the second source image and then computes the logical And of the result with the first source image, on a pixel-by-pixel basis. It uses the following equation:

$$\text{dst}[x][y][i] = \text{src1}[x][y][i] \ \& \ (\sim \text{src2}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.
src1 Pointer to first source image.
src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAndNot1_Inp\(3MLIB\)](#), [mllib_ImageAndNot2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAutoCorrel – auto-correlation of an image

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageAutoCorrel(mllib_d64 *correl, const mllib_image *img,
 mllib_s32 dx, mllib_s32 dy);`

Description The `mllib_ImageAutoCorrel()` function computes the auto-correlation of an image, given an offset.

It uses the following equation:

$$\text{correl}[i] = \frac{1}{(w-dx)*(h-dy)} * \sum_{x=0}^{w-dx-1} \sum_{y=0}^{h-dy-1} (\text{img}[x][y][i] * \text{img}[x+dx][y+dy][i])$$

where *w* and *h* are the width and height of the image, respectively.

Parameters The function takes the following arguments:

- correl* Pointer to auto-correlation array where size is equal to the number of channels. `correl[i]` contains the auto-correlation of channel *i*.
- img* Pointer to image.
- dx* Displacement in the X direction.
- dy* Displacement in the Y direction.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAutoCorrel_Fp\(3MLIB\)](#), [mllib_ImageCrossCorrel\(3MLIB\)](#), [mllib_ImageCrossCorrel_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAutoCorrel_Fp – auto-correlation of an image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageAutoCorrel_Fp(mllib_d64 *correl,
    const mllib_image *img, mllib_s32 dx, mllib_s32 dy);
```

Description The `mllib_ImageAutoCorrel_Fp()` function computes the auto-correlation of a floating-point image, given an offset.

It uses the following equation:

$$\text{correl}[i] = \frac{1}{(w-dx)*(h-dy)} * \sum_{x=0}^{w-dx-1} \sum_{y=0}^{h-dy-1} (\text{img}[x][y][i] * \text{img}[x+dx][y+dy][i])$$

where `w` and `h` are the width and height of the image, respectively.

Parameters The function takes the following arguments:

- correl* Pointer to auto-correlation array where size is equal to the number of channels. `correl[i]` contains the auto-correlation of channel `i`.
- img* Pointer to image.
- dx* Displacement in the X direction.
- dy* Displacement in the Y direction.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAutoCorrel\(3MLIB\)](#), [mllib_ImageCrossCorrel\(3MLIB\)](#), [mllib_ImageCrossCorrel_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAve – average of two images

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageAve(mllib_image *dst, const mllib_image *src1,
    const mllib_image *src2);
```

Description The mllib_ImageAve() function computes the average of two images on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = (src1[x][y][i] + src2[x][y][i] + 1) / 2$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to first source image.
- src2* Pointer to second source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAve_Fp\(3MLIB\)](#), [mllib_ImageAve_Fp_Inp\(3MLIB\)](#), [mllib_ImageAve_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAve_Fp – average of two images

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAve_Fp(mllib_image *dst, const mllib_image *src1,
                               const mllib_image *src2);
```

Description The `mllib_ImageAve_Fp()` function computes the average of two floating-point images on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = (src1[x][y][i] + src2[x][y][i]) / 2$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src1 Pointer to first source image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAve\(3MLIB\)](#), [mllib_ImageAve_Fp_Inp\(3MLIB\)](#), [mllib_ImageAve_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAve_Fp_Inp – average of two images, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageAve_Fp_Inp(mllib_image *src1dst,
    const mllib_image *src2);
```

Description The `mllib_ImageAve_Fp_Inp()` function computes the average of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = (\text{src1dst}[x][y][i] + \text{src2}[x][y][i]) / 2$$

Parameters The function takes the following arguments:

- src1dst* Pointer to first source and destination image.
- src2* Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAve\(3MLIB\)](#), [mllib_ImageAve_Fp\(3MLIB\)](#), [mllib_ImageAve_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageAve_Inp – average of two images, in place

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageAve_Inp(mllib_image *src1dst, const mllib_image *src2);
```

Description The `mllib_ImageAve_Inp()` function computes the average of two images on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = (\text{src1dst}[x][y][i] + \text{src2}[x][y][i] + 1) / 2$$

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAve\(3MLIB\)](#), [mllib_ImageAve_Fp\(3MLIB\)](#), [mllib_ImageAve_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlend1_Fp_Inp – blend with an alpha image

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageBlend1_Fp_Inp(mllib_image *src1dst,  
    const mllib_image *src2, const mllib_image *alpha);
```

Description The `mllib_ImageBlend1_Fp_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{src1dst}[x][y][i] = \text{alpha}[x][y][0] * \text{src1dst}[x][y][i] + \\ (1 - \text{alpha}[x][y][0]) * \text{src2}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{src1dst}[x][y][i] = \text{alpha}[x][y][i] * \text{src1dst}[x][y][i] + \\ (1 - \text{alpha}[x][y][i]) * \text{src2}[x][y][i]$$

Parameters The function takes the following arguments:

- src1dst* Pointer to first source and destination image.
- src2* Pointer to second source image.
- alpha* Alpha image used to control blending. The pixels in this image should have values in the range of [0.0,1.0].

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend_Fp\(3MLIB\)](#), [mllib_ImageBlend1_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlend1_Inp – blend with an alpha image, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageBlend1_Inp(mllib_image *src1dst,
    const mllib_image *src2, const mllib_image *alpha);
```

Description The `mllib_ImageBlend1_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{src1dst}[x][y][i] = a[x][y][0] * \text{src1dst}[x][y][i] + (1 - a[x][y][0]) * \text{src2}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{src1dst}[x][y][i] = a[x][y][i] * \text{src1dst}[x][y][i] + (1 - a[x][y][i]) * \text{src2}[x][y][i]$$

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

alpha Alpha image used to control blending. The *a* value equals ($\text{alpha} * 2^{**(-8)}$) for MLIB_BYTE image, ($\text{alpha} * 2^{**(-15)}$) for MLIB_SHORT image, ($\text{alpha} * 2^{**(-16)}$) for MLIB_USHORT image, and ($\text{alpha} * 2^{**(-31)}$) for MLIB_INT image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend_Fp\(3MLIB\)](#), [mllib_ImageBlend1_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlend2_Fp_Inp – blend with an alpha image

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageBlend2_Fp_Inp(mllib_image *src2dst,  
    const mllib_image *src1, const mllib_image *alpha);
```

Description The `mllib_ImageBlend2_Fp_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when a is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{src2dst}[x][y][i] = \text{alpha}[x][y][0] * \text{src1}[x][y][i] + (1 - \text{alpha}[x][y][0]) * \text{src2dst}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{src2dst}[x][y][i] = \text{alpha}[x][y][i] * \text{src1}[x][y][i] + (1 - \text{alpha}[x][y][i]) * \text{src2dst}[x][y][i]$$

It uses the following equation:

Parameters The function takes the following arguments:

- src2dst* Pointer to second source and destination image.
- src1* Pointer to first source image.
- alpha* Alpha image used to control blending. The pixels in this image should have values in the range of [0.0, 1.0].

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend_Fp\(3MLIB\)](#),
[mllib_ImageBlend1_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend1_Inp\(3MLIB\)](#),
[mllib_ImageBlend2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlend2_Inp – blend with an alpha image, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageBlend2_Inp(mllib_image *src2dst,
    const mllib_image *src1, const mllib_image *alpha);
```

Description The `mllib_ImageBlend2_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{src2dst}[x][y][i] = a[x][y][0] * \text{src1}[x][y][i] + (1 - a[x][y][0]) * \text{src2dst}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{src2dst}[x][y][i] = a[x][y][i] * \text{src1}[x][y][i] + (1 - a[x][y][i]) * \text{src2dst}[x][y][i]$$

Parameters The function takes the following arguments:

src2dst Pointer to second source and destination image.

src1 Pointer to first source image.

alpha Alpha image used to control blending. The *a* value equals ($\text{alpha} * 2^{**}(-8)$) for MLIB_BYTE image, ($\text{alpha} * 2^{**}(-15)$) for MLIB_SHORT image, ($\text{alpha} * 2^{**}(-16)$) for MLIB_USHORT image, and ($\text{alpha} * 2^{**}(-31)$) for MLIB_INT image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend_Fp\(3MLIB\)](#), [mllib_ImageBlend1_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend1_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlend – blend with an alpha image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageBlend(mllib_image *dst, const mllib_image *src1,
                             const mllib_image *src2, const mllib_image *alpha);
```

Description The `mllib_ImageBlend()` function blends two images together on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{dst}[x][y][i] = a[x][y][0] * \text{src1}[x][y][i] + (1 - a[x][y][0]) * \text{src2}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{dst}[x][y][i] = a[x][y][i] * \text{src1}[x][y][i] + (1 - a[x][y][i]) * \text{src2}[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to first source image.
- src2* Pointer to second source image.
- alpha* Alpha image used to control blending. The *a* value equals $(\text{alpha} * 2^{**(-8)})$ for `MLIB_BYTE` image, $(\text{alpha} * 2^{**(-15)})$ for `MLIB_SHORT` image, $(\text{alpha} * 2^{**(-16)})$ for `MLIB_USHORT` image, and $(\text{alpha} * 2^{**(-31)})$ for `MLIB_INT` image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend_Fp\(3MLIB\)](#), [mllib_ImageBlend1_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend1_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageBlend_BSRC1_BSRC2, mlib_ImageBlend_DA_DA, mlib_ImageBlend_DA_DC, mlib_ImageBlend_DA_OMDA, mlib_ImageBlend_DA_OMDC, mlib_ImageBlend_DA_OMSA, mlib_ImageBlend_DA_ONE, mlib_ImageBlend_DA_SA, mlib_ImageBlend_DA_SAS, mlib_ImageBlend_DA_ZERO, mlib_ImageBlend_OMDA_DA, mlib_ImageBlend_OMDA_DC, mlib_ImageBlend_OMDA_OMDA, mlib_ImageBlend_OMDA_OMDC, mlib_ImageBlend_OMDA_OMSA, mlib_ImageBlend_OMDA_ONE, mlib_ImageBlend_OMDA_SA, mlib_ImageBlend_OMDA_SAS, mlib_ImageBlend_OMDA_ZERO, mlib_ImageBlend_OMSA_DA, mlib_ImageBlend_OMSA_DC, mlib_ImageBlend_OMSA_OMDA, mlib_ImageBlend_OMSA_OMDC, mlib_ImageBlend_OMSA_OMSA, mlib_ImageBlend_OMSA_ONE, mlib_ImageBlend_OMSA_SA, mlib_ImageBlend_OMSA_SAS, mlib_ImageBlend_OMSA_ZERO, mlib_ImageBlend_OMSC_DA, mlib_ImageBlend_OMSC_DC, mlib_ImageBlend_OMSC_OMDA, mlib_ImageBlend_OMSC_OMDC, mlib_ImageBlend_OMSC_OMSA, mlib_ImageBlend_OMSC_ONE, mlib_ImageBlend_OMSC_SA, mlib_ImageBlend_OMSC_SAS, mlib_ImageBlend_OMSC_ZERO, mlib_ImageBlend_ONE_DA, mlib_ImageBlend_ONE_DC, mlib_ImageBlend_ONE_OMDA, mlib_ImageBlend_ONE_OMDC, mlib_ImageBlend_ONE_OMSA, mlib_ImageBlend_ONE_ONE, mlib_ImageBlend_ONE_SA, mlib_ImageBlend_ONE_SAS, mlib_ImageBlend_ONE_ZERO, mlib_ImageBlend_SA_DA, mlib_ImageBlend_SA_DC, mlib_ImageBlend_SA_OMDA, mlib_ImageBlend_SA_OMDC, mlib_ImageBlend_SA_OMSA, mlib_ImageBlend_SA_ONE, mlib_ImageBlend_SA_SA, mlib_ImageBlend_SA_SAS, mlib_ImageBlend_SA_ZERO, mlib_ImageBlend_SC_DA, mlib_ImageBlend_SC_DC, mlib_ImageBlend_SC_OMDA, mlib_ImageBlend_SC_OMDC, mlib_ImageBlend_SC_OMSA, mlib_ImageBlend_SC_ONE, mlib_ImageBlend_SC_SA, mlib_ImageBlend_SC_SAS, mlib_ImageBlend_SC_ZERO, mlib_ImageBlend_ZERO_DA, mlib_ImageBlend_ZERO_DC, mlib_ImageBlend_ZERO_OMDA, mlib_ImageBlend_ZERO_OMDC, mlib_ImageBlend_ZERO_OMSA, mlib_ImageBlend_ZERO_ONE, mlib_ImageBlend_ZERO_SA, mlib_ImageBlend_ZERO_SAS, mlib_ImageBlend_ZERO_ZERO – blending

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend_BSRC1_BSRC2(mlib_image *dst,
    const mlib_image *src1, const mlib_image *src2, mlib_s32 cmask);
```

Description This group of functions supports digital image composition. They are low-level, non-in-place, blending functions.

The image type must be MLIB_BYTE. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

BSRC1 is one of the following: ZERO, ONE, SC, OMSC, DA, SA, OMDA, or OMSA. BSRC2 is one of the following: ZERO, ONE, DC, OMDC, DA, SA, OMDA, OMSA, or SAS.

The following are predefined blend factor types used in mediaLib image composition functions.

```
/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mlib_blend;
```

See the following table for the definitions of the blend factors.

Type	Blend Factor [*]	Abbr.
MLIB_BLEND_ZERO	(0,0,0,0)	ZERO
MLIB_BLEND_ONE	(1,1,1,1)	ONE
MLIB_BLEND_DST_COLOR	(Rd,Gd,Bd,Ad)	DC
MLIB_BLEND_SRC_COLOR	(Rs,Gs,Bs,As)	SC
MLIB_BLEND_ONE_MINUS_DST_COLOR	(1,1,1,1)-(Rd,Gd,Bd,Ad)	OMDC
MLIB_BLEND_ONE_MINUS_SRC_COLOR	(1,1,1,1)-(Rs,Gs,Bs,As)	OMSC
MLIB_BLEND_DST_ALPHA	(Ad,Ad,Ad,Ad)	DA
MLIB_BLEND_SRC_ALPHA	(As,As,As,As)	SA
MLIB_BLEND_ONE_MINUS_DST_ALPHA	(1,1,1,1)-(Ad,Ad,Ad,Ad)	OMDA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA	(1,1,1,1)-(As,As,As,As)	OMSA
MLIB_BLEND_SRC_ALPHA_SATURATE	(f,f,f,1)	SAS

[*]: The components of the first source image pixel are (Rd,Gd,Bd,Ad), and the components of the second source pixel are (Rs,Gs,Bs,As). Function $f = \min(As, 1 - Ad)$.

The blending formula for non-in-place processing is:

$$Cd = Cs1*S1 + Cs2*S2$$

where Cd is the destination pixel (Rd,Gd,Bd,Ad), Cs1 is the first source pixel (Rs1,Gs1,Bs1,As1), Cs2 is the second source pixel (Rs2,Gs2,Bs2,As2), and S1 and S2 are the blend factors for the first and second sources, respectively.

Parameters Each of the functions takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to the first source image.
- src2* Pointer to the second source image.
- cmask* Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. cmask must be either 0x01 or 0x08.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageBlend_BSRC1_BSRC2_Inp\(3MLIB\)](#), [mlib_ImageComposite\(3MLIB\)](#), [mlib_ImageComposite_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageBlend_BSRC1_BSRC2_Inp, mlib_ImageBlend_DA_DA_Inp, mlib_ImageBlend_DA_DC_Inp, mlib_ImageBlend_DA_OMDA_Inp, mlib_ImageBlend_DA_OMDC_Inp, mlib_ImageBlend_DA_OMSA_Inp, mlib_ImageBlend_DA_ONE_Inp, mlib_ImageBlend_DA_SA_Inp, mlib_ImageBlend_DA_SAS_Inp, mlib_ImageBlend_DA_ZERO_Inp, mlib_ImageBlend_OMDA_DA_Inp, mlib_ImageBlend_OMDA_DC_Inp, mlib_ImageBlend_OMDA_OMDA_Inp, mlib_ImageBlend_OMDA_OMDC_Inp, mlib_ImageBlend_OMDA_OMSA_Inp, mlib_ImageBlend_OMDA_ONE_Inp, mlib_ImageBlend_OMDA_SA_Inp, mlib_ImageBlend_OMDA_SAS_Inp, mlib_ImageBlend_OMDA_ZERO_Inp, mlib_ImageBlend_OMSA_DA_Inp, mlib_ImageBlend_OMSA_DC_Inp, mlib_ImageBlend_OMSA_OMDA_Inp, mlib_ImageBlend_OMSA_OMDC_Inp, mlib_ImageBlend_OMSA_OMSA_Inp, mlib_ImageBlend_OMSA_ONE_Inp, mlib_ImageBlend_OMSA_SA_Inp, mlib_ImageBlend_OMSA_SAS_Inp, mlib_ImageBlend_OMSA_ZERO_Inp, mlib_ImageBlend_OMSC_DA_Inp, mlib_ImageBlend_OMSC_DC_Inp, mlib_ImageBlend_OMSC_OMDA_Inp, mlib_ImageBlend_OMSC_OMDC_Inp, mlib_ImageBlend_OMSC_OMSA_Inp, mlib_ImageBlend_OMSC_ONE_Inp, mlib_ImageBlend_OMSC_SA_Inp, mlib_ImageBlend_OMSC_SAS_Inp, mlib_ImageBlend_OMSC_ZERO_Inp, mlib_ImageBlend_ONE_DA_Inp, mlib_ImageBlend_ONE_DC_Inp, mlib_ImageBlend_ONE_OMDA_Inp, mlib_ImageBlend_ONE_OMDC_Inp, mlib_ImageBlend_ONE_OMSA_Inp, mlib_ImageBlend_ONE_ONE_Inp, mlib_ImageBlend_ONE_SA_Inp, mlib_ImageBlend_ONE_SAS_Inp, mlib_ImageBlend_ONE_ZERO_Inp, mlib_ImageBlend_SA_DA_Inp, mlib_ImageBlend_SA_DC_Inp, mlib_ImageBlend_SA_OMDA_Inp, mlib_ImageBlend_SA_OMDC_Inp, mlib_ImageBlend_SA_OMSA_Inp, mlib_ImageBlend_SA_ONE_Inp, mlib_ImageBlend_SA_SA_Inp, mlib_ImageBlend_SA_SAS_Inp, mlib_ImageBlend_SA_ZERO_Inp, mlib_ImageBlend_SC_DA_Inp, mlib_ImageBlend_SC_DC_Inp, mlib_ImageBlend_SC_OMDA_Inp, mlib_ImageBlend_SC_OMDC_Inp, mlib_ImageBlend_SC_OMSA_Inp, mlib_ImageBlend_SC_ONE_Inp, mlib_ImageBlend_SC_SA_Inp, mlib_ImageBlend_SC_SAS_Inp, mlib_ImageBlend_SC_ZERO_Inp, mlib_ImageBlend_ZERO_DA_Inp, mlib_ImageBlend_ZERO_DC_Inp, mlib_ImageBlend_ZERO_OMDA_Inp, mlib_ImageBlend_ZERO_OMDC_Inp, mlib_ImageBlend_ZERO_OMSA_Inp, mlib_ImageBlend_ZERO_ONE_Inp, mlib_ImageBlend_ZERO_SA_Inp, mlib_ImageBlend_ZERO_SAS_Inp, mlib_ImageBlend_ZERO_ZERO_Inp – blending, in place

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageBlend_BSRC1_BSRC2_Inp(mlib_image *src1dst,  
      const mlib_image *src2, mlib_s32 cmask);
```

Description This group of functions supports digital image composition. They are low-level, in-place, blending functions.

The image type must be `MLIB_BYTE`. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

BSRC1 is one of the following: ZERO, ONE, SC, OMSC, DA, SA, OMDA, or OMSA. BSRC2 is one of the following: ZERO, ONE, DC, OMDC, DA, SA, OMDA, OMSA, or SAS.

The following are predefined blend factor types used in mediaLib image composition functions.

```
/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mLib_blend;
```

See the following table for the definitions of the blend factors.

Type	Blend Factor [*]	Abbr.
<code>MLIB_BLEND_ZERO</code>	(0,0,0,0)	ZERO
<code>MLIB_BLEND_ONE</code>	(1,1,1,1)	ONE
<code>MLIB_BLEND_DST_COLOR</code>	(Rd,Gd,Bd,Ad)	DC
<code>MLIB_BLEND_SRC_COLOR</code>	(Rs,Gs,Bs,As)	SC
<code>MLIB_BLEND_ONE_MINUS_DST_COLOR</code>	(1,1,1,1)-(Rd,Gd,Bd,Ad)	OMDC
<code>MLIB_BLEND_ONE_MINUS_SRC_COLOR</code>	(1,1,1,1)-(Rs,Gs,Bs,As)	OMSC
<code>MLIB_BLEND_DST_ALPHA</code>	(Ad,Ad,Ad,Ad)	DA
<code>MLIB_BLEND_SRC_ALPHA</code>	(As,As,As,As)	SA
<code>MLIB_BLEND_ONE_MINUS_DST_ALPHA</code>	(1,1,1,1)-(Ad,Ad,Ad,Ad)	OMDA
<code>MLIB_BLEND_ONE_MINUS_SRC_ALPHA</code>	(1,1,1,1)-(As,As,As,As)	OMSA
<code>MLIB_BLEND_SRC_ALPHA_SATURATE</code>	(f,f,f,1)	SAS

[*]: The components of the first source image pixel are (Rd,Gd,Bd,Ad), and the components of the second source pixel are (Rs,Gs,Bs,As). Function $f = \min(As, 1 - Ad)$. The first source image is also the destination image.

The blending formula for in-place processing is:

$$C_d = C_d * D + C_s * S$$

where C_d is the destination pixel (Rd,Gd,Bd,Ad), C_s is the source pixel (Rs,Gs,Bs,As), and D and S are the blend factors for the destination and source, respectively.

Parameters Each of the functions takes the following arguments:

- src1dst* Pointer to the first source and the destination image.
- src2* Pointer to the second source image.
- cmask* Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. cmask must be either 0x01 or 0x08.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTEVALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend_BSRC1_BSRC2\(3MLIB\)](#), [mllib_ImageComposite\(3MLIB\)](#), [mllib_ImageComposite_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendColor – blend an image and a color

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageBlendColor(mllib_image *dst, const mllib_image *src,
                                   const mlib_s32 *color, mlib_s32 cmask);
```

Description The `mllib_ImageBlendColor()` function blends an image and a color with the alpha channel.

It uses the following equation:

$$C_d = C_s * A_s + C_c * (1 - A_s)$$

$$A_d = 1.0$$

where, C_s and C_d are the RGB color components of the source and destination images, respectively. A_s and A_d are the alpha components of the source and destination images, respectively. C_c is the color component of the constant color.

For `MLIB_BYTE` images, the alpha coefficients are in Q8 format. For `MLIB_SHORT` images, the alpha coefficients are in Q15 format and must be positive. For `MLIB_USHORT` images, the alpha coefficients are in Q16 format. For `MLIB_INT` images, the alpha coefficients are in Q31 format and must be positive.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

color Array of constant color components.

cmask Channel mask to indicate the alpha channel. Each bit of *cmask* represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageBlendColor_Inp(3MLIB)`, `mllib_ImageBlendColor_Fp(3MLIB)`,
`mllib_ImageBlendColor_Fp_Inp(3MLIB)`, `attributes(5)`

Name mllib_ImageBlendColor_Fp – blend an image and a color

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageBlendColor_Fp(mllib_image *dst, const mllib_image *src,  
    const mlib_d64 *color, mlib_s32 cmask);
```

Description The `mllib_ImageBlendColor_Fp()` function blends an image and a color with the alpha channel.

It uses the following equation:

$$C_d = C_s * A_s + C_c * (1 - A_s)$$
$$A_d = 1.0$$

where, C_s and C_d are the RGB color components of the source and destination images, respectively. A_s and A_d are the alpha components of the source and destination images, respectively. C_c is the color component of the constant color.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the alpha coefficients are assumed to be in the range of $[0.0, 1.0]$.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- color* Array of constant color components.
- cmask* Channel mask to indicate the alpha channel. Each bit of *cmask* represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendColor_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlendColor\(3MLIB\)](#),
[mllib_ImageBlendColor_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendColor_Fp_Inp – blend an image and a color, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageBlendColor_Fp_Inp(mllib_image *srcdst,
      const mllib_d64 *color, mllib_s32 cmask);
```

Description The `mllib_ImageBlendColor_Fp_Inp()` function blends an image and a color with the alpha channel.

It uses the following equation:

$$C_d = C_s * A_s + C_c * (1 - A_s)$$

$$A_d = 1.0$$

where, C_s and C_d are the RGB color components of the source and destination images, respectively. A_s and A_d are the alpha components of the source and destination images, respectively. C_c is the color component of the constant color.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the alpha coefficients are assumed to be in the range of $[0.0, 1.0]$.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

Parameters The function takes the following arguments:

srcdst Pointer to the source and destination image.

color Array of constant color components.

cmask Channel mask to indicate the alpha channel. Each bit of *cmask* represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendColor_Fp\(3MLIB\)](#), [mllib_ImageBlendColor\(3MLIB\)](#), [mllib_ImageBlendColor_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendColor_Inp – blend an image and a color, in place

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageBlendColor_Inp(mllib_image *srcdst,  
    const mlib_s32 *color, mlib_s32 cmask);
```

Description The `mllib_ImageBlendColor_Inp()` function blends an image and a color with the alpha channel.

It uses the following equation:

$C_d = C_s * A_s + C_c * (1 - A_s)$
 $A_d = 1.0$

where, C_s and C_d are the RGB color components of the source and destination images, respectively. A_s and A_d are the alpha components of the source and destination images, respectively. C_c is the color component of the constant color.

For `MLIB_BYTE` images, the alpha coefficients are in Q8 format. For `MLIB_SHORT` images, the alpha coefficients are in Q15 format and must be positive. For `MLIB_USHORT` images, the alpha coefficients are in Q16 format. For `MLIB_INT` images, the alpha coefficients are in Q31 format and must be positive.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

Parameters The function takes the following arguments:

srcdst Pointer to the source and destination image.

color Array of constant color components.

cmask Channel mask to indicate the alpha channel. Each bit of *cmask* represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageBlendColor(3MLIB)`, `mlib_ImageBlendColor_Fp(3MLIB)`,
`mlib_ImageBlendColor_Fp_Inp(3MLIB)`, `attributes(5)`

Name mllib_ImageBlend_Fp – blend with an alpha image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageBlend_Fp(mllib_image *dst, const mllib_image *src1,
    const mllib_image *src2, const mllib_image *alpha);
```

Description The `mllib_ImageBlend_Fp()` function blends two images together on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

$$\text{dst}[x][y][i] = \text{alpha}[x][y][0] * \text{src1}[x][y][i] + (1 - \text{alpha}[x][y][0]) * \text{src2}[x][y][i]$$

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

$$\text{dst}[x][y][i] = \text{alpha}[x][y][i] * \text{src1}[x][y][i] + (1 - \text{alpha}[x][y][i]) * \text{src2}[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to first source image.
- src2* Pointer to second source image.
- alpha* Alpha image used to control blending. The pixels in this image should have values in the range of [0.0, 1.0].

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend1_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend1_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Fp_Inp\(3MLIB\)](#), [mllib_ImageBlend2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendMulti – blend multiple images

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageBlendMulti(mllib_image *dst, const mllib_image **srcs,  
    const mllib_image **alphas, const mllib_s32 *c, mllib_s32 n);
```

Description The `mllib_ImageBlendMulti()` function blends multiple source images, using multiple alpha images, into a single destination image.

All images involved should have the same data type and same size and the source and destination images should have the same number of channels. The alpha images should have either 1 channel or the same number of channels as the sources and destination. A single-channel alpha image would be applied to all channels of the corresponding source image. Single and multi-channel alpha images can be mixed in the same invocation.

It uses the following equation:

$$dst[x][y][i] = \frac{\sum_{k=0}^{n-1} \{alphas[k][x][y][j] * srcs[k][x][y][i]\}}{\sum_{k=0}^{n-1} \{alphas[k][x][y][j]\}}$$

or

$$dst[x][y][i] = c[i] \quad \text{if } \sum_{k=0}^{n-1} \{alphas[k][x][y][j]\} = 0$$

where $j = i$ for multi-channel alpha images; $j = 0$ for single-channel alpha images.

Parameters The function takes the following arguments:

dst Pointer to destination image.
srcs Pointer to an array of source images.
alphas Pointer to an array of alpha images.
c Background color.
n Number of source images to be blended.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendMulti_Fp\(3MLIB\)](#), [mllib_ImageBlend\(3MLIB\)](#),
[mllib_ImageBlend_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendMulti_Fp – blend multiple images

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageBlendMulti_Fp(mllib_image *dst,  
    const mllib_image **srcs, const mllib_image **alphas, const mllib_d64 *c,  
    mllib_s32 n);
```

Description The `mllib_ImageBlendMulti_Fp()` function blends multiple source images, using multiple alpha images, into a single destination image.

All images involved should have the same data type and same size and the source and destination images should have the same number of channels. The alpha images should have either 1 channel or the same number of channels as the sources and destination. A single-channel alpha image would be applied to all channels of the corresponding source image. Single and multi-channel alpha images can be mixed in the same invocation.

It uses the following equation:

$$\text{dst}[x][y][i] = \frac{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j] * \text{srcs}[k][x][y][i]\}}{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j]\}}$$

or

$$\text{dst}[x][y][i] = c[i] \quad \text{if } \sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j]\} = 0$$

where $j = i$ for multi-channel alpha images; $j = 0$ for single-channel alpha images.

Parameters The function takes the following arguments:

dst Pointer to destination image.
srcs Pointer to an array of source images.
alphas Pointer to an array of alpha images.
c Background color.
n Number of source images to be blended.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendMulti\(3MLIB\)](#), [mllib_ImageBlend\(3MLIB\)](#), [mllib_ImageBlend_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendRGBA2ARGB – image blending and channel reordering

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageBlendRGBA2ARGB(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageBlendRGBA2ARGB()` function blends the source image of the RGBA format into the destination image of the ARGB format.

The image type must be `MLIB_BYTE`. The source and destination images must contain four channels.

It uses the following equation:

```
Cd = Cs*As + Cd*(1 - As)
Ad = Ad
```

where, Cs and Cd are the RGB color components of the source and destination images, respectively. As and Ad are the alpha components of the source and destination images, respectively.

Parameters The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendRGBA2BGRA\(3MLIB\)](#), [mllib_ImageBlend_OMSA_SA_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageBlendRGBA2BGRA – image blending and channel reordering

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageBlendRGBA2BGRA(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageBlendRGBA2BGRA()` function blends the source image of the RGBA format into the destination image of the BGRA format.

The image type must be `MLIB_BYTE`. The source and destination images must contain four channels.

It uses the following equation:

$$C_d = C_s * A_s + C_d * (1 - A_s)$$
$$A_d = A_d$$

where, C_s and C_d are the RGB color components of the source and destination images, respectively. A_s and A_d are the alpha components of the source and destination images, respectively.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlendRGBA2ARGB\(3MLIB\)](#), [mllib_ImageBlend_OMSA_SA_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageChannelCopy – channel copy

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageChannelCopy(mllib_image *dst, const mllib_image *src,  
                                   mllib_s32 cmask);
```

Description The `mllib_ImageChannelCopy()` function copies the selected channels of the source image into the corresponding channels of the destination image. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

- dst* Pointer to a destination image.
- src* Pointer to a source image.
- cmask* Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageChannelExtract\(3MLIB\)](#), [mllib_ImageChannelInsert\(3MLIB\)](#), [mllib_ImageChannelMerge\(3MLIB\)](#), [mllib_ImageChannelSplit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageChannelExtract – channel extract

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageChannelExtract(mllib_image *dst,
    const mllib_image *src, mllib_s32 cmask);
```

Description In the `mllib_ImageChannelExtract()` function, the selected N channels in the source image are copied into the destination image, where N is the number of channels in the destination image. If more than N channels are selected, then the leftmost N channels are extracted. If less than N channels are selected, then the function returns failure status. The channel mask is defined with respect to the source image. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

- dst* Pointer to a destination image.
- src* Pointer to a source image.
- cmask* Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageChannelCopy\(3MLIB\)](#), [mllib_ImageChannelInsert\(3MLIB\)](#), [mllib_ImageChannelMerge\(3MLIB\)](#), [mllib_ImageChannelSplit\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageChannelInsert – channel insert

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```

```
mlib_status mlib_ImageChannelInsert(mlib_image *dst, const mlib_image *src,
    mlib_s32 cmask);
```

Description In the `mlib_ImageChannelInsert()` function, all *N* channels in the source image are copied into the selected channels in the destination image, where *N* is the number of channels in the source image. If more than *N* channels are selected, then the leftmost *N* channels are inserted. If less than *N* channels are selected, then the function returns failure status. The channel mask is defined with respect to the destination image. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

dst Pointer to a destination image.

src Pointer to a source image.

cmask Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageChannelCopy\(3MLIB\)](#), [mlib_ImageChannelExtract\(3MLIB\)](#), [mlib_ImageChannelMerge\(3MLIB\)](#), [mlib_ImageChannelSplit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageChannelMerge – channel merge

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageChannelMerge(mllib_image *dst,  
                                     const mllib_image **srcs);
```

Description The `mllib_ImageChannelMerge()` function converts an array of single-channel images into a multi-channel image.

```
A0 A1 A2 ...  
B0 B1 B2 ... ==> A0 B0 C0 A1 B1 C1 A2 B2 C2 ...  
C0 C1 C2 ...
```

All images must have the same type, same width, and same height. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`. The destination image must have the number of channels equal to the number of images in the `srcs` array. The source images must be single-channel images.

Parameters The function takes the following arguments:

dst Pointer to a multi-channel destination image.

srcs Pointer to an array of single-channel source images.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageChannelCopy\(3MLIB\)](#), [mllib_ImageChannelExtract\(3MLIB\)](#),
[mllib_ImageChannelInsert\(3MLIB\)](#), [mllib_ImageChannelSplit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageChannelSplit – channel split

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageChannelSplit(mllib_image **dsts,
                                     const mllib_image *src);
```

Description The `mllib_ImageChannelSplit()` function converts a multi-channel image into an array of single-channel images.

```

                                     A0 A1 A2 ...
A0 B0 C0 A1 B1 C1 A2 B2 C2 ...  ==> B0 B1 B2 ...
                                     C0 C1 C2 ...
```

All images must have the same type, same width, and same height. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`. The source image must have the number of channels equal to the number of images in the *dsts* array. The destination images must be single-channel images.

Parameters The function takes the following arguments:

dsts Pointer to an array of single-channel destination images.

src Pointer to a multi-channel source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageChannelCopy\(3MLIB\)](#), [mllib_ImageChannelExtract\(3MLIB\)](#), [mllib_ImageChannelInsert\(3MLIB\)](#), [mllib_ImageChannelMerge\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageClear – clear

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageClear(mllib_image *img, const mllib_s32 *color);`

Description The `mllib_ImageClear()` function sets an image to a specific color. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

It uses the following equation:

`img[x][y][i] = color[i]`

Parameters The function takes the following arguments:

img Pointer to an image.

color Array of color values by channel. `color[i]` contains the value for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageClear_Fp\(3MLIB\)](#), [mllib_ImageClearEdge\(3MLIB\)](#),
[mllib_ImageClearEdge_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageClearEdge – sets edges of an image to a specific color

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```

```
mlib_status mlib_ImageClearEdge(mlib_image *img, mlib_s32 dx, mlib_s32 dy,
    const mlib_s32 *color);
```

Description The `mlib_ImageClearEdge()` function sets edges of an image to a specific color. This function can be used in conjunction with the convolve and other spatial functions to fill in the pixel values along the edges. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

img Pointer to an image.

dx Number of columns on the left and right edges of the image to be cleared.

dy Number of rows at the top and bottom edges of the image to be cleared.

color Array of color values by channel. `color[i]` contains the value for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageClear\(3MLIB\)](#), [mlib_ImageClear_Fp\(3MLIB\)](#),
[mlib_ImageClearEdge_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageClearEdge_Fp – sets edges of an image to a specific color

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageClearEdge_Fp(mllib_image *img, mllib_s32 dx,  
                                     mllib_s32 dy, const mllib_d64 *color);
```

Description The `mllib_ImageClearEdge_Fp()` function sets edges of an image to a specific color. This function can be used in conjunction with the convolve and other spatial functions to fill in the pixel values along the edges. The data type of the image can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

- img* Pointer to an image.
- dx* Number of columns on the left and right edges of the image to be cleared.
- dy* Number of rows at the top and bottom edges of the image to be cleared.
- color* Array of color values by channel. `color[i]` contains the value for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageClear\(3MLIB\)](#), [mllib_ImageClearEdge\(3MLIB\)](#),
[mllib_ImageClearEdge_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageClear_Fp – clear

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>
```

```
mllib_status mllib_ImageClear_Fp(mllib_image *img, const mllib_d64 *color);
```

Description The `mllib_ImageClear_Fp()` function sets an image to a specific color. The data type of the image can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

It uses the following equation:

```
img[x][y][i] = color[i]
```

Parameters The function takes the following arguments:

img Pointer to an image.

color Array of color values by channel. `color[i]` contains the value for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageClear\(3MLIB\)](#), [mllib_ImageClearEdge\(3MLIB\)](#),
[mllib_ImageClearEdge_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorConvert1 – color conversion using a 3x3 floating-point matrix

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorConvert1(mllib_image *dst, const mllib_image *src,  
                                     const mllib_d64 *cmat);
```

Description The `mllib_ImageColorConvert1()` function takes a 3x3 floating-point conversion matrix and converts the source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{aligned} |dst[x][y][0]| &= |cmat[0] \ cmat[1] \ cmat[2]| * |src[x][y][0]| \\ |dst[x][y][1]| &= |cmat[3] \ cmat[4] \ cmat[5]| * |src[x][y][1]| \\ |dst[x][y][2]| &= |cmat[6] \ cmat[7] \ cmat[8]| * |src[x][y][2]| \end{aligned}$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmat* Conversion matrix in row major order.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1_Fp\(3MLIB\)](#), [mllib_ImageColorConvert2\(3MLIB\)](#),
[mllib_ImageColorConvert2_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ\(3MLIB\)](#),
[mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB\(3MLIB\)](#),
[mllib_ImageColorXYZ2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorConvert1_Fp – color conversion using a 3x3 floating-point matrix

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorConvert1_Fp(mllib_image *dst,
    const mllib_image *src, const mllib_d64 *cmat);
```

Description The `mllib_ImageColorConvert1_Fp()` function takes a 3x3 floating point conversion matrix and converts the floating-point source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{aligned} |dst[x][y][0]| & \quad |cmat[0] \quad cmat[1] \quad cmat[2]| & \quad |src[x][y][0]| \\ |dst[x][y][1]| &= |cmat[3] \quad cmat[4] \quad cmat[5]| * & \quad |src[x][y][1]| \\ |dst[x][y][2]| & \quad |cmat[6] \quad cmat[7] \quad cmat[8]| & \quad |src[x][y][2]| \end{aligned}$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmat* Conversion matrix in row major order.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1\(3MLIB\)](#), [mllib_ImageColorConvert2\(3MLIB\)](#), [mllib_ImageColorConvert2_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorConvert2 – color conversion using a 3x3 floating-point matrix and a three-element offset

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorConvert2(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *cmat, const mllib_d64 *offset);
```

Description The `mllib_ImageColorConvert2()` function takes a 3x3 floating-point conversion matrix and a three-element offset and converts the source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{array}{l} |dst[x][y][0]| \quad |cmat[0] \ cmat[1] \ cmat[2]| \quad |src[x][y][0]| \quad |offset[0]| \\ |dst[x][y][1]| = |cmat[3] \ cmat[4] \ cmat[5]| * |src[x][y][1]| + |offset[1]| \\ |dst[x][y][2]| \quad |cmat[6] \ cmat[7] \ cmat[8]| \quad |src[x][y][2]| \quad |offset[2]| \end{array}$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmat* Conversion matrix in row major order.
- offset* Offset array.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1\(3MLIB\)](#), [mllib_ImageColorConvert1_Fp\(3MLIB\)](#),
[mllib_ImageColorConvert2_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2YCC\(3MLIB\)](#),
[mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#), [mllib_ImageColorYCC2RGB\(3MLIB\)](#),
[mllib_ImageColorYCC2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorConvert2_Fp – color conversion using a 3x3 floating-point matrix and a three-element offset

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorConvert2_Fp(mllib_image *dst,
    const mllib_image *src, const mllib_d64 *cmat,
    const mllib_d64 *offset);
```

Description The mllib_ImageColorConvert2_Fp() function takes a 3x3 floating-point conversion matrix and a three-element offset and converts the floating-point source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{aligned} |dst[x][y][0]| & \quad |cmat[0] \ cmat[1] \ cmat[2]| \quad |src[x][y][0]| \quad |offset[0]| \\ |dst[x][y][1]| & = |cmat[3] \ cmat[4] \ cmat[5]| * |src[x][y][1]| + |offset[1]| \\ |dst[x][y][2]| & \quad |cmat[6] \ cmat[7] \ cmat[8]| \quad |src[x][y][2]| \quad |offset[2]| \end{aligned}$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmat* Conversion matrix in row major order.
- offset* Offset array.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1\(3MLIB\)](#), [mllib_ImageColorConvert1_Fp\(3MLIB\)](#), [mllib_ImageColorConvert2\(3MLIB\)](#), [mllib_ImageColorRGB2YCC\(3MLIB\)](#), [mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#), [mllib_ImageColorYCC2RGB\(3MLIB\)](#), [mllib_ImageColorYCC2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorDitherFree – release the internal data structure for image dithering

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`void mllib_ImageColorDitherFree(void *colormap);`

Description The `mllib_ImageColorDitherFree()` function releases an internal data structure, *colormap*, which was created by `mllib_ImageColorDitherInit()` and was used by one of the following functions for image dithering:

`mllib_ImageColorErrorDiffusion3x3`
`mllib_ImageColorErrorDiffusionMxN`
`mllib_ImageColorOrderedDither8x8`
`mllib_ImageColorOrderedDitherMxN`

Parameters The function takes the following arguments:

colormap Internal data structure for image dithering.

Return Values None.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorDitherInit\(3MLIB\)](#), [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#), [mllib_ImageColorErrorDiffusionMxN\(3MLIB\)](#), [mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorOrderedDitherMxN\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorDitherInit – initialization for image dithering

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorDitherInit(void **colormap, const mllib_s32 *dimensions,
                                         mllib_type intype, mllib_type outtype, mllib_s32 channels, mllib_s32 entries,
                                         mllib_s32 offset, void **lut);
```

Description The `mllib_ImageColorDitherInit()` function creates an internal data structure, *colormap*, which can be used by one of the following functions for image dithering:

```
mllib_ImageColorErrorDiffusion3x3
mllib_ImageColorErrorDiffusionMxN
mllib_ImageColorOrderedDither8x8
mllib_ImageColorOrderedDitherMxN
```

The *lut* might have either 1 or 3 channels. The type of the *lut* can be one of the following:

```
MLIB_BYTE in, MLIB_BYTE out (i.e., BYTE-to-BYTE)
MLIB_BIT in, MLIB_BYTE out (i.e., BIT-to-BYTE)
```

If *dimensions* == NULL, then no colorcube will be created. In this case, the user-provided lookup table, *lut*, will be used for dithering.

If *dimensions* != NULL, then a colorcube is created from scratch in a way shown in the following example.

To dither an RGB image of type MLIB_BYTE to a color-indexed image of type MLIB_BYTE, we can use the following parameters:

```
mllib_s32 dimensions[] = {2, 3, 4};
mllib_type intype = MLIB_BYTE;
mllib_type outtype = MLIB_BYTE;
mllib_s32 channels = 3;
mllib_s32 offset = 6;
```

These values would lead to the creation of a colorcube that would dither red values in the source image to one of 2 red levels, green values to one of 3 green levels, and blue values to one of 4 blue levels. You could picture this colorcube as a cube with dimensions of 2, 3, and 4. The index values assigned to the elements in that cube can be described by the following lookup table:

Indexes	Red Values	Green Values	Blue Values
0			
...			
5			

Indexes	Red Values	Green Values	Blue Values
6	0	0	0
7	255	0	0
8	0	128	0
9	255	128	0
10	0	255	0
11	255	255	0
12	0	0	85
13	255	0	85
14	0	128	85
15	255	128	85
16	0	255	85
17	255	255	85
18	0	0	170
19	255	0	170
20	0	128	170
21	255	128	170
22	0	255	170
23	255	255	170
24	0	0	255
25	255	0	255
26	0	128	255
27	255	128	255
28	0	255	255
29	255	255	255
...			

The distance between level changes in each channel of the lookup table is determined by the following formulas:

```

multipliers[0] = signof(dimensions[0])*1;
multipliers[i] = signof(dimensions[i])*
                abs(multipliers[i-1]*dimension[i-1]);

```

A negative `dimensions[i]`, so as to a negative `multipliers[i]`, indicates that the values in a color ramp for channel `i` should appear in decreasing as opposed to increasing order.

For each channel `i`, the values of the levels are determined by the following formulas:

```

double delta = (dataMax - dataMin)/(abs(dimensions[i]) - 1);
int levels[j] = (int)(j*delta + 0.5);

```

where `dataMax` and `dataMin` are the maximum and minimum values, respectively, for data type *inttype*.

Whenever a colorcube is created, if `lut != NULL`, the lookup table will be filled according to the colorcube and supplied parameters like *offset*. For the example shown above, the lookup table will start from line 6. In this case, it is the user's responsibility to allocate memory for the lookup table.

Parameters The function takes the following arguments:

<i>colormap</i>	Internal data structure for image dithering.
<i>dimensions</i>	Dimensions of the colorcube in the colormap structure.
<i>inttype</i>	Data type of the source image and the lookup table.
<i>outtype</i>	Data type of the destination indexed image.
<i>channels</i>	Number of channels of the lookup table and source image.
<i>entries</i>	Number of entries of the lookup table.
<i>offset</i>	Index offset of the lookup table.
<i>lut</i>	Lookup table.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorDitherFree\(3MLIB\)](#), [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#), [mllib_ImageColorErrorDiffusionMxN\(3MLIB\)](#), [mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorOrderedDitherMxN\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorErrorDiffusion3x3 – true color to indexed color conversion using error diffusion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorErrorDiffusion3x3(mllib_image *dst,  
        const mllib_image *src, const mllib_s32 *kernel, mllib_s32 scale,  
        const void *colormap);
```

Description The `mllib_ImageColorErrorDiffusion3x3()` function converts a true color image to a pseudo color image with the method of error diffusion dithering. The source image can be an `MLIB_BYTE` or `MLIB_SHORT` image with three or four channels. The destination must be a single-channel `MLIB_BYTE` or `MLIB_SHORT` image.

The last parameter, *colormap*, is an internal data structure for inverse color mapping. Create it by calling the `mllib_ImageColorTrue2IndexInit()` function.

Parameters The function takes the following arguments:

- dst* Pointer to destination or destination image.
- src* Pointer to source or source image.
- kernel* Pointer to the 3x3 error-distribution kernel, in row major order.
- scale* The scaling factor for kernel to convert the input integer coefficients into floating-point coefficients:
- `floating-point coefficient = integer coefficient * \`
`2**(-scale)`
- colormap* Internal data structure for inverse color mapping.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorTrue2Index\(3MLIB\)](#), [mllib_ImageColorTrue2IndexFree\(3MLIB\)](#), [mllib_ImageColorTrue2IndexInit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorErrorDiffusionMxN – true-color to indexed-color or grayscale to black-white conversion, using error diffusion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorErrorDiffusionMxN(mllib_image *dst,
        const mllib_image *src, const mllib_s32 *kernel,
        mllib_s32 m, mllib_s32 n, mllib_s32 dm, mllib_s32 dn,
        mllib_s32 scale, const void *colormap);
```

Description The `mllib_ImageColorErrorDiffusionMxN()` function converts a 3-channel image to a 1-channel indexed image, or converts a 1-channel grayscale image to a 1-channel `MLIB_BIT` image, with the method of error diffusion.

The *src* can be an `MLIB_BYTE` image with 1 or 3 channels. The *dst* must be a 1-channel `MLIB_BIT` or `MLIB_BYTE` image.

The *colormap* must be created by `mllib_ImageColorDitherInit()`. It may or may not have a colorcube included. If it does, the colorcube is used. Otherwise, the general lookup table included in *colormap* is used.

The kernel is required to have the following property:

```
kernel[0] = kernel[1] = ... = kernel[m*dn+dm] = 0;
kernel[m*dn+dm+1] + ... + kernel[m*n-1] = 2**scale;
scale ≥ 0
```

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>kernel</i>	Pointer to the error-distribution kernel, in row major order.
<i>m</i>	Kernel width. $m > 1$.
<i>n</i>	Kernel height. $n > 1$.
<i>dm</i>	X coordinate of the key element in the kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the kernel. $0 \leq dn < n$.
<i>scale</i>	The scaling factor for kernel to convert the input integer coefficients into floating-point coefficients: floating-point coefficient = integer coefficient * $\backslash 2^{**(-scale)}$
<i>colormap</i>	Internal data structure for image dithering.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorDitherInit\(3MLIB\)](#), [mllib_ImageColorDitherFree\(3MLIB\)](#),
[mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#),
[mllib_ImageColorOrderedDither8x8\(3MLIB\)](#),
[mllib_ImageColorOrderedDitherMxN\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorHSL2RGB – HSL to RGB color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageColorHSL2RGB(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorHSL2RGB()` function performs a conversion from hue/saturation/lightness to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

$$\begin{aligned} L' &= L & \text{if } L \leq 1/2 \\ L' &= 1 - L & \text{if } L > 1/2 \end{aligned}$$

$$\begin{aligned} V &= L + S * L' \\ P &= L - S * L' \\ Q &= L + S * L' * (1 - 2 * \text{fraction}(H * 6)) \\ T &= L - S * L' * (1 - 2 * \text{fraction}(H * 6)) \end{aligned}$$

$$\begin{aligned} R, G, B &= V, T, P & \text{if } 0 \leq H < 1/6 \\ R, G, B &= Q, V, P & \text{if } 1/6 \leq H < 2/6 \\ R, G, B &= P, V, T & \text{if } 2/6 \leq H < 3/6 \\ R, G, B &= P, Q, V & \text{if } 3/6 \leq H < 4/6 \\ R, G, B &= T, P, V & \text{if } 4/6 \leq H < 5/6 \\ R, G, B &= V, P, Q & \text{if } 5/6 \leq H < 1 \end{aligned}$$

where $0 \leq H < 1$ and $0 \leq S, L, L', V, P, Q, T, R, G, B \leq 1$.

Assuming a pixel in the source image is (h, s, l) and its corresponding pixel in the destination image is (r, g, b), then for MLIB_BYTE images, the following applies:

$$\begin{aligned} H &= h/256 \\ S &= s/255 \\ L &= l/255 \\ r &= R * 255 \\ g &= G * 255 \\ b &= B * 255 \end{aligned}$$

for MLIB_SHORT images, the following applies:

$$\begin{aligned} H &= (h + 32768)/65536 \\ S &= (s + 32768)/65535 \\ L &= (l + 32768)/65535 \\ r &= R * 65535 - 32768 \\ g &= G * 65535 - 32768 \\ b &= B * 65535 - 32768 \end{aligned}$$

for MLIB_USHORT images, the following applies:

```
H = h/65536
S = s/65535
L = l/65535
r = R*65535
g = G*65535
b = B*65535
```

and for MLIB_INT images, the following applies:

```
H = (h + 2147483648)/4294967296
S = (s + 2147483648)/4294967295
L = (l + 2147483648)/4294967295
r = R*4294967295 - 2147483648
g = G*4294967295 - 2147483648
b = B*4294967295 - 2147483648
```

Parameters The function takes the following arguments:

```
dst    Pointer to destination image.
src    Pointer to source image.
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSL2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2HSL\(3MLIB\)](#), [mllib_ImageColorRGB2HSL_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorHSL2RGB_Fp – HSL to RGB color conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageColorHSL2RGB_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageColorHSL2RGB_Fp()` function performs a conversion from hue/saturation/lightness to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

$$\begin{aligned} L' &= L && \text{if } L \leq 1/2 \\ L' &= 1 - L && \text{if } L > 1/2 \\ \\ V &= L + S * L' \\ P &= L - S * L' \\ Q &= L + S * L' * (1 - 2 * \text{fraction}(H * 6)) \\ T &= L - S * L' * (1 - 2 * \text{fraction}(H * 6)) \\ \\ R, G, B &= V, T, P && \text{if } 0 \leq H < 1/6 \\ R, G, B &= Q, V, P && \text{if } 1/6 \leq H < 2/6 \\ R, G, B &= P, V, T && \text{if } 2/6 \leq H < 3/6 \\ R, G, B &= P, Q, V && \text{if } 3/6 \leq H < 4/6 \\ R, G, B &= T, P, V && \text{if } 4/6 \leq H < 5/6 \\ R, G, B &= V, P, Q && \text{if } 5/6 \leq H < 1 \end{aligned}$$

where $0 \leq H < 1$ and $0 \leq S, L, L', V, P, Q, T, R, G, B \leq 1$.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input `H` component values must be limited to the `[0.0, 1.0]` range. Input `S` and `L` component values must be limited to the `[0.0, 1.0]` range.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSL2RGB\(3MLIB\)](#), [mllib_ImageColorRGB2HSL\(3MLIB\)](#),
[mllib_ImageColorRGB2HSL_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorHSV2RGB – HSV to RGB color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorHSV2RGB(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageColorHSV2RGB()` function performs a conversion from hue/saturation/value to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

$$\begin{aligned} P &= V(1 - S) \\ Q &= V(1 - S \cdot \text{fraction}(H \cdot 6)) \\ T &= V(1 - S(1 - \text{fraction}(H \cdot 6))) \end{aligned}$$

$$\begin{aligned} R, G, B &= V, T, P && \text{if } 0 \leq H < 1/6 \\ R, G, B &= Q, V, P && \text{if } 1/6 \leq H < 2/6 \\ R, G, B &= P, V, T && \text{if } 2/6 \leq H < 3/6 \\ R, G, B &= P, Q, V && \text{if } 3/6 \leq H < 4/6 \\ R, G, B &= T, P, V && \text{if } 4/6 \leq H < 5/6 \\ R, G, B &= V, P, Q && \text{if } 5/6 \leq H < 1 \end{aligned}$$

where $0 \leq H < 1$ and $0 \leq S, V, P, Q, T, R, G, B \leq 1$.

Assuming a pixel in the source image is (h, s, v) and its corresponding pixel in the destination image is (r, g, b) , then for `MLIB_BYTE` images, the following applies:

$$\begin{aligned} H &= h/256 \\ S &= s/255 \\ V &= v/255 \\ r &= R \cdot 255 \\ g &= G \cdot 255 \\ b &= B \cdot 255 \end{aligned}$$

for `MLIB_SHORT` images, the following applies:

$$\begin{aligned} H &= (h + 32768)/65536 \\ S &= (s + 32768)/65535 \\ V &= (v + 32768)/65535 \\ r &= R \cdot 65535 - 32768 \\ g &= G \cdot 65535 - 32768 \\ b &= B \cdot 65535 - 32768 \end{aligned}$$

for `MLIB_USHORT` images, the following applies:

$$\begin{aligned} H &= h/65536 \\ S &= s/65535 \\ V &= v/65535 \\ r &= R \cdot 65535 \\ g &= G \cdot 65535 \\ b &= B \cdot 65535 \end{aligned}$$

and for MLIB_INT images, the following applies:

```
H = (h + 2147483648)/4294967296
S = (s + 2147483648)/4294967295
V = (v + 2147483648)/4294967295
r = R*4294967295 - 2147483648
g = G*4294967295 - 2147483648
b = B*4294967295 - 2147483648
```

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSV2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2HSV\(3MLIB\)](#),
[mllib_ImageColorRGB2HSV_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorHSV2RGB_Fp – HSV to RGB color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorHSV2RGB_Fp(mllib_image *dst,  
    const mllib_image *src);
```

Description The `mllib_ImageColorHSV2RGB_Fp()` function performs a conversion from hue/saturation/value to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

$$\begin{aligned} P &= V * (1 - S) \\ Q &= V * (1 - S * \text{fraction}(H * 6)) \\ T &= V * (1 - S * (1 - \text{fraction}(H * 6))) \end{aligned}$$

$$\begin{aligned} R, G, B &= V, T, P && \text{if } 0 \leq H < 1/6 \\ R, G, B &= Q, V, P && \text{if } 1/6 \leq H < 2/6 \\ R, G, B &= P, V, T && \text{if } 2/6 \leq H < 3/6 \\ R, G, B &= P, Q, V && \text{if } 3/6 \leq H < 4/6 \\ R, G, B &= T, P, V && \text{if } 4/6 \leq H < 5/6 \\ R, G, B &= V, P, Q && \text{if } 5/6 \leq H < 1 \end{aligned}$$

where $0 \leq H < 1$ and $0 \leq S, V, P, Q, T, R, G, B \leq 1$.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input *H* component values must be limited to the $[0.0, 1.0]$ range. Input *S* and *V* component values must be limited to the $[0.0, 1.0]$ range.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSV2RGB\(3MLIB\)](#), [mllib_ImageColorRGB2HSV\(3MLIB\)](#),
[mllib_ImageColorRGB2HSV_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorOrderedDither8x8 – true color to indexed color conversion using ordered dithering

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status (mllib_image *dst, const mllib_image *src, const mllib_s32 *dmask,
              mllib_s32 scale, const void *colormap);
```

Description The `mllib_ImageColorOrderedDither8x8()` function converts a true color image to a pseudo color image with the method of ordered dithering. The source image can be an `MLIB_BYTE` or `MLIB_SHORT` image with three or four channels. The destination must be a single-channel `MLIB_BYTE` or `MLIB_SHORT` image.

This function works only with a colorcube, rather than a general lookup table. The last parameter, *colormap*, is an internal data structure (which may include a colorcube) for inverse color mapping. Create it by calling the `mllib_ImageColorTrue2IndexInit()` function.

Parameters The function takes the following arguments:

dst Pointer to destination or destination image.

src Pointer to source or source image.

dmask Pointer to the 8x8 dither mask, in row major order. The dither mask is transposed differently for different channels to reduce artifacts.

scale Scaling factor for dmask to convert the input integer coefficients into floating-point coefficients:

floating-point coefficient = integer coefficient * \

2**(-scale)

colormap Internal data structure for inverse color mapping.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#), [mllib_ImageColorTrue2Index\(3MLIB\)](#), [mllib_ImageColorTrue2IndexFree\(3MLIB\)](#), [mllib_ImageColorTrue2IndexInit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorOrderedDitherMxN – true-color to indexed-color or grayscale to black-white conversion, using ordered dithering

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorOrderedDitherMxN(mllib_image *dst,  
        const mllib_image *src, const mlib_s32 **dmask, mlib_s32 m,  
        mlib_s32 n, mlib_s32 scale, const void *colormap);
```

Description The `mllib_ImageColorOrderedDitherMxN()` function converts a 3-channel image to a 1-channel indexed image, or converts a 1-channel grayscale image to a 1-channel `MLIB_BIT` image, with the method of ordered dithering.

The *src* can be an `MLIB_BYTE` image with 1 or 3 channels. The *dst* must be a 1-channel `MLIB_BIT` or `MLIB_BYTE` image.

The *colormap* must be created by `mllib_ImageColorDitherInit()`, and it must have a colorcube included.

The dither masks are required to have the following property:

$$0 \leq \text{dmask}[i][j] < 2^{**}\text{scale}; \text{scale} > 0$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- dmask* Pointer to the dither masks, one per channel, in row major order.
- m* Mask width. $m > 1$.
- n* Mask height. $n > 1$.
- scale* Scaling factor for *dmask* to convert the input integer coefficients into floating-point coefficients:

$$\text{floating-point coefficient} = \text{integer coefficient} * \backslash 2^{*(-scale)}$$
- colormap* Internal data structure for image dithering.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `mllib_ImageColorDitherInit(3MLIB),mllib_ImageColorDitherFree(3MLIB),
mllib_ImageColorErrorDiffusion3x3(3MLIB),
mllib_ImageColorErrorDiffusionMxN(3MLIB),
mllib_ImageColorOrderedDither8x8(3MLIB),attributes(5)`

Name mllib_ImageColorRGB2CIEMono – RGB to monochrome conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorRGB2CIEMono(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageColorRGB2CIEMono()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

$$\begin{aligned} \text{dst}[x][y][0] = & 0.2125 * \text{src}[x][y][0] + \\ & 0.7154 * \text{src}[x][y][1] + \\ & 0.0721 * \text{src}[x][y][2] \end{aligned}$$

Parameters The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorRGB2CIEMono_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2Mono\(3MLIB\)](#), [mllib_ImageColorRGB2Mono_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2CIEMono_Fp – RGB to monochrome conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2CIEMono_Fp(mllib_image *dst,  
      const mllib_image *src);
```

Description The `mllib_ImageColorRGB2CIEMono_Fp()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

$$\begin{aligned} \text{dst}[x][y][0] = & 0.2125 * \text{src}[x][y][0] + \\ & 0.7154 * \text{src}[x][y][1] + \\ & 0.0721 * \text{src}[x][y][2] \end{aligned}$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorRGB2CIEMono\(3MLIB\)](#), [mllib_ImageColorRGB2Mono\(3MLIB\)](#), [mllib_ImageColorRGB2Mono_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2HSL – RGB to HSL color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2HSL(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageColorRGB2HSL()` function performs a conversion from red/green/blue to hue/saturation/lightness. The source and destination images must be three-channel images.

It uses the following equations:

```
V = max(R, G, B)
Vmin = min(R, G, B)

L = (V + Vmin)/2

S = (V - Vmin)/(V + Vmin)      if L ≤ 1/2
S = (V - Vmin)/(2 - V - Vmin)  if L > 1/2

H = (5.0 + (V - B)/(V - Vmin))/6  if R = V and G = Vmin
H = (1.0 - (V - G)/(V - Vmin))/6  if R = V and B = Vmin
H = (1.0 + (V - R)/(V - Vmin))/6  if G = V and B = Vmin
H = (3.0 - (V - B)/(V - Vmin))/6  if G = V and R = Vmin
H = (3.0 + (V - G)/(V - Vmin))/6  if B = V and R = Vmin
H = (5.0 - (V - R)/(V - Vmin))/6  if B = V and G = Vmin
H = 0.0                          if R = G = B
```

where $0 \leq R, G, B, V, Vmin, L, S \leq 1$ and $0 \leq H < 1$.

Assuming a pixel in the source image is (r, g, b) and its corresponding pixel in the destination image is (h, s, l) , then for `MLIB_BYTE` images, the following applies:

```
R = r/255
G = g/255
B = b/255
h = H*256
s = S*255
l = L*255
```

for `MLIB_SHORT` images, the following applies:

```
R = (r + 32768)/65535
G = (g + 32768)/65535
B = (b + 32768)/65535
h = H*65536 - 32768
s = S*65535 - 32768
l = L*65535 - 32768
```

for `MLIB_USHORT` images, the following applies:

```
R = r/65535
G = g/65535
B = b/65535
h = H*65536
s = S*65535
l = L*65535
```

and for MLIB_INT images, the following applies:

```
R = (r + 2147483648)/4294967295
G = (g + 2147483648)/4294967295
B = (b + 2147483648)/4294967295
h = H*4294967296 - 2147483648
s = S*4294967295 - 2147483648
l = L*4294967295 - 2147483648
```

Parameters The function takes the following arguments:

```
dst    Pointer to destination image.
src    Pointer to source image.
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSL2RGB\(3MLIB\)](#), [mllib_ImageColorHSL2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2HSL_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2HSL_Fp – RGB to HSL color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2HSL_Fp(mllib_image *dst,  
    const mllib_image *src);
```

Description The `mllib_ImageColorRGB2HSL_Fp()` function performs a conversion from red/green/blue to hue/saturation/lightness. The source and destination images must be three-channel images.

It uses the following equations:

```
V = max(R, G, B)
Vmin = min(R, G, B)

L = (V + Vmin)/2

S = (V - Vmin)/(V + Vmin)      if L ≤ 1/2
S = (V - Vmin)/(2 - V - Vmin)  if L > 1/2

H = (5.0 + (V - B)/(V - Vmin))/6  if R = V and G = Vmin
H = (1.0 - (V - G)/(V - Vmin))/6  if R = V and B = Vmin
H = (1.0 + (V - R)/(V - Vmin))/6  if G = V and B = Vmin
H = (3.0 - (V - B)/(V - Vmin))/6  if G = V and R = Vmin
H = (3.0 + (V - G)/(V - Vmin))/6  if B = V and R = Vmin
H = (5.0 - (V - R)/(V - Vmin))/6  if B = V and G = Vmin
H = 0.0                          if R = G = B
```

where $0 \leq R, G, B, V, Vmin, L, S \leq 1$ and $0 \leq H < 1$.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input `R`, `G`, and `B` component values must be limited to the `[0.0, 1.0]` range.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageColorHSL2RGB\(3MLIB\)](#), [mlib_ImageColorHSL2RGB_Fp\(3MLIB\)](#),
[mlib_ImageColorRGB2HSL\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2HSV – RGB to HSV color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageColorRGB2HSV(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorRGB2HSV()` function performs a conversion from red/green/blue to hue/saturation/value. The source and destination images must be three-channel images.

It uses the following equations:

$$\begin{aligned} V &= \max(R, G, B) \\ V_{\min} &= \min(R, G, B) \\ S &= (V - V_{\min})/V \\ H &= (5.0 + (V - B)/(V - V_{\min}))/6 \quad \text{if } R = V \text{ and } G = V_{\min} \\ H &= (1.0 - (V - G)/(V - V_{\min}))/6 \quad \text{if } R = V \text{ and } B = V_{\min} \\ H &= (1.0 + (V - R)/(V - V_{\min}))/6 \quad \text{if } G = V \text{ and } B = V_{\min} \\ H &= (3.0 - (V - B)/(V - V_{\min}))/6 \quad \text{if } G = V \text{ and } R = V_{\min} \\ H &= (3.0 + (V - G)/(V - V_{\min}))/6 \quad \text{if } B = V \text{ and } R = V_{\min} \\ H &= (5.0 - (V - R)/(V - V_{\min}))/6 \quad \text{if } B = V \text{ and } G = V_{\min} \\ H &= 0.0 \quad \quad \quad \quad \quad \quad \quad \text{if } R = G = B \end{aligned}$$

where $0 \leq R, G, B, V, V_{\min}, S \leq 1$ and $0 \leq H < 1$.

Assuming a pixel in the source image is (r, g, b) and its corresponding pixel in the destination image is (h, s, v) , then for `MLIB_BYTE` images, the following applies:

$$\begin{aligned} R &= r/255 \\ G &= g/255 \\ B &= b/255 \\ h &= H*256 \\ s &= S*255 \\ v &= V*255 \end{aligned}$$

for `MLIB_SHORT` images, the following applies:

$$\begin{aligned} R &= (r + 32768)/65535 \\ G &= (g + 32768)/65535 \\ B &= (b + 32768)/65535 \\ h &= H*65536 - 32768 \\ s &= S*65535 - 32768 \\ v &= V*65535 - 32768 \end{aligned}$$

for `MLIB_USHORT` images, the following applies:

$$\begin{aligned} R &= r/65535 \\ G &= g/65535 \\ B &= b/65535 \end{aligned}$$


```
h = H*65536
s = S*65535
v = V*65535
```

and for MLIB_INT images, the following applies:

```
R = (r + 2147483648)/4294967295
G = (g + 2147483648)/4294967295
B = (b + 2147483648)/4294967295
h = H*4294967296 - 2147483648
s = S*4294967295 - 2147483648
v = V*4294967295 - 2147483648
```

Parameters The function takes the following arguments:

```
dst    Pointer to destination image.
src    Pointer to source image.
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorHSV2RGB\(3MLIB\)](#), [mllib_ImageColorHSV2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2HSV_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2HSV_Fp – RGB to HSV color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2HSV_Fp(mllib_image *dst,  
    const mllib_image *src);
```

Description The `mllib_ImageColorRGB2HSV_Fp()` function performs a conversion from red/green/blue to hue/saturation/value. The source and destination images must be three-channel images.

It uses the following equations:

```
V = max(R, G, B)
Vmin = min(R, G, B)

S = (V - Vmin)/V

H = (5.0 + (V - B)/(V - Vmin))/6  if R = V and G = Vmin
H = (1.0 - (V - G)/(V - Vmin))/6  if R = V and B = Vmin
H = (1.0 + (V - R)/(V - Vmin))/6  if G = V and B = Vmin
H = (3.0 - (V - B)/(V - Vmin))/6  if G = V and R = Vmin
H = (3.0 + (V - G)/(V - Vmin))/6  if B = V and R = Vmin
H = (5.0 - (V - R)/(V - Vmin))/6  if B = V and G = Vmin
H = 0.0                          if R = G = B
```

where $0 \leq R, G, B, V, Vmin, S \leq 1$ and $0 \leq H < 1$.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input `R`, `G`, and `B` component values must be limited to the `[0.0, 1.0]` range.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageColorHSV2RGB\(3MLIB\)](#), [mlib_ImageColorHSV2RGB_Fp\(3MLIB\)](#),
[mlib_ImageColorRGB2HSV\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2Mono – RGB to monochrome conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2Mono(mllib_image *dst, const mllib_image *src,  
                                       const mllib_d64 *weight);
```

Description The `mllib_ImageColorRGB2Mono()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{weight}[0] * \text{src}[x][y][0] + \\ \text{weight}[1] * \text{src}[x][y][1] + \\ \text{weight}[2] * \text{src}[x][y][2]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

weight Array of three blending coefficients. It is recommended that these sum to 1.0, but it is not required.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorRGB2CIEMono\(3MLIB\)](#), [mllib_ImageColorRGB2CIEMono_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2Mono_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2Mono_Fp – RGB to monochrome conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorRGB2Mono_Fp(mllib_image *dst,  
    const mllib_image *src, const mllib_d64 *weight);
```

Description The `mllib_ImageColorRGB2Mono_Fp()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{weight}[0] * \text{src}[x][y][0] + \\ \text{weight}[1] * \text{src}[x][y][1] + \\ \text{weight}[2] * \text{src}[x][y][2]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- weight* Array of three blending coefficients. It is recommended that these sum to 1.0, but it is not required.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorRGB2CIEMono\(3MLIB\)](#), [mllib_ImageColorRGB2CIEMono_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2Mono\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorRGB2XYZ – RGB to XYZ color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageColorRGB2XYZ(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorRGB2XYZ()` function performs a color space conversion from ITU-R Rec.708 RGB with D64 white point to CIE XYZ.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} |X| \\ |Y| \\ |Z| \end{bmatrix} = \begin{bmatrix} |cmat[0] & cmat[1] & cmat[2]| \\ |cmat[3] & cmat[4] & cmat[5]| \\ |cmat[6] & cmat[7] & cmat[8]| \end{bmatrix} * \begin{bmatrix} |R| \\ |G| \\ |B| \end{bmatrix}$$

where

```
cmat[] = { 0.412453, 0.357580, 0.180423,  
           0.212671, 0.715160, 0.072169,  
           0.019334, 0.119193, 0.950227 };  
src[x][y] = { R, G, B };  
dst[x][y] = { X, Y, Z };
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1\(3MLIB\)](#), [mllib_ImageColorConvert1_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorYCC2RGB\(3MLIB\)](#), [mllib_ImageColorYCC2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name

mllib_ImageColorRGB2XYZ_Fp – RGB to XYZ color conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorRGB2XYZ_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description

The `mllib_ImageColorRGB2XYZ_Fp()` function performs a color space conversion from ITU-R Rec.708 RGB with D64 white point to CIE XYZ.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \text{cmat}[0] & \text{cmat}[1] & \text{cmat}[2] \\ \text{cmat}[3] & \text{cmat}[4] & \text{cmat}[5] \\ \text{cmat}[6] & \text{cmat}[7] & \text{cmat}[8] \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where

```
cmat[] = { 0.412453, 0.357580, 0.180423,
           0.212671, 0.715160, 0.072169,
           0.019334, 0.119193, 0.950227 };
src[x][y] = { R, G, B };
dst[x][y] = { X, Y, Z };
```

Parameters

The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also

`mllib_ImageColorConvert1(3MLIB)`, `mllib_ImageColorConvert1_Fp(3MLIB)`,
`mllib_ImageColorRGB2XYZ(3MLIB)`, `mllib_ImageColorXYZ2RGB(3MLIB)`,
`mllib_ImageColorXYZ2RGB_Fp(3MLIB)`, `mllib_ImageColorYCC2RGB(3MLIB)`,
`mllib_ImageColorYCC2RGB_Fp(3MLIB)`, `mllib_ImageColorYCC2RGB_Fp(3MLIB)`,
[attributes\(5\)](#)

Name mllib_ImageColorRGB2YCC – RGB to YCC color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageColorRGB2YCC(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorRGB2YCC()` function performs a color space conversion from computer R'G'B' to ITU-R Rec.601 Y'CbCr.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{array}{l} |Y'| \quad |cmat[0] \ cmat[1] \ cmat[2]| \quad |R'| \quad |offset[0]| \\ |Cb| = |cmat[3] \ cmat[4] \ cmat[5]| * |G'| + |offset[1]| \\ |Cr| \quad |cmat[6] \ cmat[7] \ cmat[8]| \quad |B'| \quad |offset[2]| \end{array}$$

where

```
cmat[] = { 65.738/256, 129.057/256, 25.064/256,
           -37.945/256, -74.494/256, 112.439/256,
           112.439/256, -94.154/256, -18.285/256 };
offset[] = { 16, 128, 128 };
src[x][y] = { R', G', B' };
dst[x][y] = { Y', Cb, Cr };
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert2\(3MLIB\)](#), [mllib_ImageColorConvert2_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#), [mllib_ImageColorXYZ2RGB\(3MLIB\)](#),
[mllib_ImageColorXYZ2RGB_Fp\(3MLIB\)](#), [mllib_ImageColorYCC2RGB\(3MLIB\)](#),
[mllib_ImageColorYCC2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name

mllib_ImageColorRGB2YCC_Fp – RGB to YCC color conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorRGB2YCC_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description

The `mllib_ImageColorRGB2YCC_Fp()` function performs a color space conversion from computer R'G'B' to ITU-R Rec.601 Y'CbCr.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} cmat[0] & cmat[1] & cmat[2] \\ cmat[3] & cmat[4] & cmat[5] \\ cmat[6] & cmat[7] & cmat[8] \end{bmatrix} * \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} offset[0] \\ offset[1] \\ offset[2] \end{bmatrix}$$

where

```
cmat[] = { 65.738/256, 129.057/256, 25.064/256,
           -37.945/256, -74.494/256, 112.439/256,
           112.439/256, -94.154/256, -18.285/256 };
offset[] = { 16, 128, 128 };
src[x][y] = { R', G', B' };
dst[x][y] = { Y', Cb, Cr };
```

Parameters

The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also

`mllib_ImageColorConvert2(3MLIB)`, `mllib_ImageColorConvert2_Fp(3MLIB)`,
`mllib_ImageColorRGB2YCC(3MLIB)`, `mllib_ImageColorXYZ2RGB(3MLIB)`,
`mllib_ImageColorXYZ2RGB_Fp(3MLIB)`, `mllib_ImageColorYCC2RGB(3MLIB)`,
`mllib_ImageColorYCC2RGB_Fp(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageColorTrue2Index – true color to indexed color using nearest matched LUT entries

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageColorTrue2Index(mllib_image *dst,  
    const mllib_image *src, const void *colormap);
```

Description The `mllib_ImageColorTrue2Index()` function converts a true color image to a pseudo color image with the method of finding the nearest matched lookup table entry for each pixel. The source image can be an `MLIB_BYTE` or `MLIB_SHORT` image with three or four channels. The destination must be a single-channel `MLIB_BYTE` or `MLIB_SHORT` image.

The last parameter, *colormap*, is an internal data structure (which includes the lookup table) for inverse color mapping. Create it by calling the `mllib_ImageColorTrue2IndexInit()` function.

Parameters The function takes the following arguments:

- dst* Pointer to destination or destination image.
- src* Pointer to source or source image.
- colormap* Internal data structure for inverse color mapping.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#),
[mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorTrue2IndexFree\(3MLIB\)](#),
[mllib_ImageColorTrue2IndexInit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorTrue2IndexFree – releases the internal data structure for true color to indexed color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`void mllib_ImageColorTrue2IndexFree(void *colormap);`

Description The `mllib_ImageColorTrue2IndexFree()` function releases the internal data structure, *colormap*, which was created by `mllib_ImageColorTrue2IndexInit()` and was used by one of the following functions:

`mllib_ImageColorTrue2Index`
`mllib_ImageColorErrorDiffusion3x3`
`mllib_ImageColorOrderedDither8x8`

Parameters The function takes the following arguments:

colormap Internal data structure for inverse color mapping.

Return Values None.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#),
[mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorTrue2Index\(3MLIB\)](#),
[mllib_ImageColorTrue2IndexInit\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorTrue2IndexInit – initialization for true color to indexed color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageColorTrue2IndexInit(void **colormap, mllib_s32 bits,  
                                             mllib_type intype, mllib_type outtype, mllib_s32 channels,  
                                             mllib_s32 entries, mllib_s32 offset, const void **table);
```

Description The `mllib_ImageColorTrue2IndexInit()` function creates and initializes an internal data structure based on the input lookup table and other parameters for inverse color mapping.

The lookup table can have either three or four channels. The number of channels of the lookup table should match that of the source image provided to the function that will use the *colormap* structure created by this function.

The type of the lookup table can be one of the following:

```
MLIB_BYTE in, MLIB_BYTE out (i.e., BYTE-to-BYTE)  
MLIB_SHORT in, MLIB_SHORT out (i.e., SHORT-to-SHORT)  
MLIB_SHORT in, MLIB_BYTE out (i.e., SHORT-to-BYTE)
```

The input type of the lookup table should match the type of the destination image; the output type of the lookup table should match the source image type. The source and destination images are the images provided to the function that is going to use the *colormap* structure created by `mllib_ImageColorTrue2IndexInit()` to do inverse color mapping.

Parameters The function takes the following arguments:

<i>colormap</i>	Internal data structure for inverse color mapping.
<i>bits</i>	Number of bits per color component used in the colorcube of the colormap structure. (If <code>bits = 0</code> , then no colorcube is created. But the inverse color mapping might be done by using the original lookup table.)
<i>intype</i>	Data type of the source image and lookup table.
<i>outtype</i>	Data type of the destination indexed image.
<i>channels</i>	Number of channels of the lookup table.
<i>entries</i>	Number of entries of the lookup table.
<i>offset</i>	The first entry offset of the lookup table.
<i>table</i>	The lookup table (LUT).

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorErrorDiffusion3x3\(3MLIB\)](#),
[mllib_ImageColorOrderedDither8x8\(3MLIB\)](#), [mllib_ImageColorTrue2Index\(3MLIB\)](#),
[mllib_ImageColorTrue2IndexFree\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorXYZ2RGB – XYZ to RGB color conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageColorXYZ2RGB(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorXYZ2RGB()` function performs a color space conversion from CIE XYZ to ITU-R Rec.708 RGB with D64 white point.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{aligned} |R| &= |cmat[0] \ cmat[1] \ cmat[2]| \ |X| \\ |G| &= |cmat[3] \ cmat[4] \ cmat[5]| \ * \ |Y| \\ |B| &= |cmat[6] \ cmat[7] \ cmat[8]| \ |Z| \end{aligned}$$

where

```
cmat[] = { 3.240479, -1.537150, -0.498535,  
          -0.969256,  1.875992,  0.041566,  
          0.055648, -0.204043,  1.057311 };  
src[x][y] = { X, Y, Z };  
dst[x][y] = { R, G, B };
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert1\(3MLIB\)](#), [mllib_ImageColorConvert1_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2XYZ\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2YCC\(3MLIB\)](#), [mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#),
[mllib_ImageColorXYZ2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageColorXYZ2RGB_Fp – XYZ to RGB color conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorXYZ2RGB_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageColorXYZ2RGB_Fp()` function performs a color space conversion from CIE XYZ to ITU-R Rec.708 RGB with D64 white point.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} \text{cmat}[0] & \text{cmat}[1] & \text{cmat}[2] \\ \text{cmat}[3] & \text{cmat}[4] & \text{cmat}[5] \\ \text{cmat}[6] & \text{cmat}[7] & \text{cmat}[8] \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

where

```
cmat[] = { 3.240479, -1.537150, -0.498535,
           -0.969256,  1.875992,  0.041566,
           0.055648, -0.204043,  1.057311 };
src[x][y] = { X, Y, Z };
dst[x][y] = { R, G, B };
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageColorConvert1(3MLIB)`, `mllib_ImageColorConvert1_Fp(3MLIB)`, `mllib_ImageColorRGB2XYZ(3MLIB)`, `mllib_ImageColorRGB2XYZ_Fp(3MLIB)`, `mllib_ImageColorRGB2YCC(3MLIB)`, `mllib_ImageColorRGB2YCC_Fp(3MLIB)`, `mllib_ImageColorXYZ2RGB(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageColorYCC2RGB – YCC to RGB color conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageColorYCC2RGB(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageColorYCC2RGB()` function performs a color space conversion from ITU-R Rec.601 Y'CbCr to computer R'G'B'.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} \text{cmat}[0] & \text{cmat}[1] & \text{cmat}[2] \\ \text{cmat}[3] & \text{cmat}[4] & \text{cmat}[5] \\ \text{cmat}[6] & \text{cmat}[7] & \text{cmat}[8] \end{bmatrix} \begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} + \begin{bmatrix} \text{offset}[0] \\ \text{offset}[1] \\ \text{offset}[2] \end{bmatrix}$$

where

```
cmat[] = { 298.082/256, 0.000/256, 408.583/256,
           298.082/256, -100.291/256, -208.120/256,
           298.082/256, 516.411/256, 0.000/256 };
offset[] = { -222.922, 135.575, -276.836 };
src[x][y] = { Y', Cb, Cr };
dst[x][y] = { R', G', B' };
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageColorConvert2\(3MLIB\)](#), [mllib_ImageColorConvert2_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ\(3MLIB\)](#), [mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#), [mllib_ImageColorRGB2YCC\(3MLIB\)](#), [mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#), [mllib_ImageColorYCC2RGB_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name

mllib_ImageColorYCC2RGB_Fp – YCC to RGB color conversion

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageColorYCC2RGB_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description

The `mllib_ImageColorYCC2RGB_Fp()` function performs a color space conversion from ITU-R Rec.601 Y'CbCr to computer R'G'B'.

The source and destination images must be three-channel images.

It uses the following equation:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} \text{cmat}[0] & \text{cmat}[1] & \text{cmat}[2] \\ \text{cmat}[3] & \text{cmat}[4] & \text{cmat}[5] \\ \text{cmat}[6] & \text{cmat}[7] & \text{cmat}[8] \end{bmatrix} * \begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} + \begin{bmatrix} \text{offset}[0] \\ \text{offset}[1] \\ \text{offset}[2] \end{bmatrix}$$

where

```
cmat[] = { 298.082/256,    0.000/256,  408.583/256,
           298.082/256, -100.291/256, -208.120/256,
           298.082/256,  516.411/256,   0.000/256 };
offset[] = { -222.922, 135.575, -276.836 };
src[x][y] = { Y', Cb, Cr };
dst[x][y] = { R', G', B' };
```

Parameters

The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also

[mllib_ImageColorConvert2\(3MLIB\)](#),
[mllib_ImageColorConvert2_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2XYZ\(3MLIB\)](#),
[mllib_ImageColorRGB2XYZ_Fp\(3MLIB\)](#),
[mllib_ImageColorRGB2YCC\(3MLIB\)](#),
[mllib_ImageColorRGB2YCC_Fp\(3MLIB\)](#),
[mllib_ImageColorYCC2RGB\(3MLIB\)](#),
[attributes\(5\)](#)

Name mllib_ImageComposite – image composition

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageComposite(mllib_image *dst, const mllib_image *src1,  
    const mllib_image *src2, mllib_blend bsrc1, mllib_blend bsrc2,  
    mllib_s32 cmask);
```

Description The `mllib_ImageComposite()` function supports digital image composition.

It is a wrapper of the `mllib_ImageBlend_BSRC1_BSRC2` group of functions and can perform various types of composition based on the parameters passed in, whereas each function in that group can perform only the one kind of composition denoted by its name.

The image type must be `MLIB_BYTE`. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

The following are predefined blend factor types used in mediaLib image composition functions.

```
/* image blend factors */  
typedef enum {  
    MLIB_BLEND_ZERO,  
    MLIB_BLEND_ONE,  
    MLIB_BLEND_DST_COLOR,  
    MLIB_BLEND_SRC_COLOR,  
    MLIB_BLEND_ONE_MINUS_DST_COLOR,  
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,  
    MLIB_BLEND_DST_ALPHA,  
    MLIB_BLEND_SRC_ALPHA,  
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,  
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,  
    MLIB_BLEND_SRC_ALPHA_SATURATE  
} mllib_blend;
```

See the following table for the definitions of the blend factors.

Type	Blend Factor [*]	Abbr.
MLIB_BLEND_ZERO	(0,0,0,0)	ZERO
MLIB_BLEND_ONE	(1,1,1,1)	ONE
MLIB_BLEND_DST_COLOR	(Rd,Gd,Bd,Ad)	DC
MLIB_BLEND_SRC_COLOR	(Rs,Gs,Bs,As)	SC
MLIB_BLEND_ONE_MINUS_DST_COLOR	(1,1,1,1)-(Rd,Gd,Bd,Ad)	OMDC
MLIB_BLEND_ONE_MINUS_SRC_COLOR	(1,1,1,1)-(Rs,Gs,Bs,As)	OMSC

Type	Blend Factor [*]	Abbr.
MLIB_BLEND_DST_ALPHA	(Ad,Ad,Ad,Ad)	DA
MLIB_BLEND_SRC_ALPHA	(As,As,As,As)	SA
MLIB_BLEND_ONE_MINUS_DST_ALPHA	(1,1,1,1)-(Ad,Ad,Ad,Ad)	OMDA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA	(1,1,1,1)-(As,As,As,As)	OMSA
MLIB_BLEND_SRC_ALPHA_SATURATE	(f,f,f,1)	SAS

[*]: The components of the first source image pixel are (Rd, Gd, Bd, Ad), and the components of the second source pixel are (Rs, Gs, Bs, As). Function $f = \min(As, 1 - Ad)$.

The blending formula for non-in-place processing is:

$$Cd = Cs1 * S1 + Cs2 * S2$$

where Cd is the destination pixel (Rd, Gd, Bd, Ad), Cs1 is the first source pixel (Rs1, Gs1, Bs1, As1), Cs2 is the second source pixel (Rs2, Gs2, Bs2, As2), and S1 and S2 are the blend factors for the first and second sources, respectively.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src1</i>	Pointer to the first source image.
<i>src2</i>	Pointer to the second source image.
<i>bsrc1</i>	Blend factor type for the first source image.
<i>bsrc2</i>	Blend factor type for the second source image.
<i>cmask</i>	Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. cmask must be either 0x01 or 0x08.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageBlend_BSRC1_BSRC2\(3MLIB\)](#), [mllib_ImageBlend_BSRC1_BSRC2_Inp\(3MLIB\)](#),
[mllib_ImageComposite_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageComposite_Inp – image composition, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageComposite_Inp(mllib_image *src1dst,
    const mllib_image *src2, mllib_blend bsrc1, mllib_blend bsrc2,
    mllib_s32 cmask);
```

Description The `mllib_ImageComposite_Inp()` function supports digital image composition. It is a wrapper of the `mllib_ImageBlend_BSRC1_BSRC2_Inp` group of functions and can perform various types of composition based on the parameters passed in, whereas each function in the `mllib_ImageBlend_BSRC1_BSRC2_Inp` group can perform only the one kind of composition denoted by its name.

The image type must be `MLIB_BYTE`. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

The following are predefined blend factor types used in mediaLib image composition functions.

```
/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mllib_blend;
```

See the following table for the definitions of the blend factors.

Type	Blend Factor [*]	Abbr.
MLIB_BLEND_ZERO	(0,0,0,0)	ZERO
MLIB_BLEND_ONE	(1,1,1,1)	ONE
MLIB_BLEND_DST_COLOR	(Rd,Gd,Bd,Ad)	DC
MLIB_BLEND_SRC_COLOR	(Rs,Gs,Bs,As)	SC
MLIB_BLEND_ONE_MINUS_DST_COLOR	(1,1,1,1)-(Rd,Gd,Bd,Ad)	OMDC

Type	Blend Factor [*]	Abbr.
MLIB_BLEND_ONE_MINUS_SRC_COLOR	(1,1,1,1)-(Rs,Gs,Bs,As)	OMSC
MLIB_BLEND_DST_ALPHA	(Ad,Ad,Ad,Ad)	DA
MLIB_BLEND_SRC_ALPHA	(As,As,As,As)	SA
MLIB_BLEND_ONE_MINUS_DST_ALPHA	(1,1,1,1)-(Ad,Ad,Ad,Ad)	OMDA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA	(1,1,1,1)-(As,As,As,As)	OMSA
MLIB_BLEND_SRC_ALPHA_SATURATE	(f,f,f,1)	SAS

[*]: The components of the first source image pixel are (Rd , Gd , Bd , Ad) , and the components of the second source pixel are (Rs , Gs , Bs , As) . Function $f = \min (As , 1 - Ad)$. The first source image is also the destination image.

The blending formula for in-place processing is:

$$Cd = Cd*D + Cs*S$$

where Cd is the destination pixel (Rd , Gd , Bd , Ad) , Cs is the source pixel (Rs , Gs , Bs , As) , and D and S are the blend factors for the destination and source, respectively.

Parameters The function takes the following arguments:

- src1dst* Pointer to the first source and the destination image.
- src2* Pointer to the second source image.
- bsrc1* Blend factor type for the first source image.
- bsrc2* Blend factor type for the second source image.
- cmask* Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. cmask must be either 0x01 or 0x08.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageBlend_BSRC1_BSRC2\(3MLIB\)](#), [mlib_ImageBlend_BSRC1_BSRC2_Inp\(3MLIB\)](#),
[mlib_ImageComposite\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAdd – addition with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstAdd(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The mllib_ImageConstAdd() function adds a constant to an image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] + src[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* An array of constants to be added to each pixel by channel. *c*[*i*] contains the constant for channel *i*.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAdd_Fp\(3MLIB\)](#), [mllib_ImageConstAdd_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAdd_Fp – addition with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstAdd_Fp(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *c);
```

Description The `mllib_ImageConstAdd_Fp()` function adds a constant to a floating-point image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] + src[x][y][i]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c An array of constants to be added to each pixel by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAdd\(3MLIB\)](#), [mllib_ImageConstAdd_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAdd_Fp_Inp – addition with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstAdd_Fp_Inp(mllib_image *srcdst,  
                                         const mllib_d64 *c);
```

Description The `mllib_ImageConstAdd_Fp_Inp()` function adds a constant to a floating-point image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] + \text{srcdst}[x][y][i]$$

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* An array of constants to be added to each pixel by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAdd\(3MLIB\)](#), [mllib_ImageConstAdd_Fp\(3MLIB\)](#),
[mllib_ImageConstAdd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAdd_Inp – addition with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstAdd_Inp(mllib_image *srcdst, const mllib_s32 *c);
```

Description The `mllib_ImageConstAdd_Inp()` function adds a constant to an image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$srcdst[x][y][i] = c[i] + srcdst[x][y][i]$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

c An array of constants to be added to each pixel by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAdd\(3MLIB\)](#), [mllib_ImageConstAdd_Fp\(3MLIB\)](#), [mllib_ImageConstAdd_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAnd – And with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstAnd(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The mllib_ImageConstAnd() function computes the And of the source image with a constant.

It uses the following equation:

$$dst[x][y][i] = c[i] \& src[x][y][i]$$

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants to be applied to each pixel. *c*[*i*] contains the constant for channel *i*.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAnd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAnd_Inp – And with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageConstAnd_Inp(mllib_image *srcdst, const mllib_s32 *c);`

Description The `mllib_ImageConstAnd_Inp()` function computes the And of the source image with a constant, in place.

It uses the following equation:

$$srcdst[x][y][i] = c[i] \& srcdst[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAnd\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAndNot – AndNot with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstAndNot(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstAndNot()` function computes the And of the Not of the source image with a constant.

It uses the following equation:

$$dst[x][y][i] = c[i] \ \& \ (\sim src[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAndNot_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstAndNot_Inp – AndNot with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstAndNot_Inp(mllib_image *srcdst,  
                                         const mlib_s32 *c);
```

Description The `mllib_ImageConstAndNot_Inp()` function computes the And of the Not of the source image with a constant, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] \ \& \ (\sim \text{srcdst}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

srcdst Pointer to first source and destination image.

c Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstAndNot\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstDiv – division into a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstDiv(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *c);
```

Description The `mllib_ImageConstDiv()` function divides each pixel in an image into a constant value on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = c[i] / \text{src}[x][y][i]$$

In the case of $\text{src}[x][y][i] = 0$,

$$\begin{aligned} \text{dst}[x][y][i] &= 0 && \text{if } c[i] = 0 \\ \text{dst}[x][y][i] &= \text{DATA_TYPE_MAX} && \text{if } c[i] > 0 \\ \text{dst}[x][y][i] &= \text{DATA_TYPE_MIN} && \text{if } c[i] < 0 \end{aligned}$$

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Constant into which each pixel is divided. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstDiv_Fp\(3MLIB\)](#), [mllib_ImageConstDiv_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstDiv_Inp\(3MLIB\)](#), [mllib_ImageConstDivShift\(3MLIB\)](#), [mllib_ImageConstDivShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstDiv_Fp – division into a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConstDiv_Fp(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *c);
```

Description The `mllib_ImageConstDiv_Fp()` function divides each pixel in a floating-point image by a constant value on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] / src[x][y][i]$$

where the operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Constant into which each pixel is divided. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConstDiv(3MLIB)`, `mllib_ImageConstDiv_Fp_Inp(3MLIB)`, `mllib_ImageConstDiv_Inp(3MLIB)`, `mllib_ImageConstDivShift(3MLIB)`, `mllib_ImageConstDivShift_Inp(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConstDiv_Fp_Inp – division into a constant

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConstDiv_Fp_Inp(mllib_image *srcdst,  
    const mllib_d64 *c);
```

Description The `mllib_ImageConstDiv_Fp_Inp()` function divides each pixel in a floating-point image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] / \text{srcdst}[x][y][i]$$

where the operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* Constant into which each pixel is divided. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstDiv\(3MLIB\)](#), [mllib_ImageConstDiv_Fp\(3MLIB\)](#),
[mllib_ImageConstDiv_Inp\(3MLIB\)](#), [mllib_ImageConstDivShift\(3MLIB\)](#),
[mllib_ImageConstDivShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstDiv_Inp – division into a constant

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageConstDiv_Inp(mllib_image *srcdst, const mllib_d64 *c);`

Description The `mllib_ImageConstDiv_Inp()` function divides each pixel in an image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] / \text{srcdst}[x][y][i]$$

In the case of `srcdst[x][y][i] = 0`,

$$\begin{aligned} \text{srcdst}[x][y][i] &= 0 && \text{if } c[i] = 0 \\ \text{srcdst}[x][y][i] &= \text{DATA_TYPE_MAX} && \text{if } c[i] > 0 \\ \text{srcdst}[x][y][i] &= \text{DATA_TYPE_MIN} && \text{if } c[i] < 0 \end{aligned}$$

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `c` Constant into which each pixel is divided. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstDiv\(3MLIB\)](#), [mllib_ImageConstDiv_Fp\(3MLIB\)](#),
[mllib_ImageConstDiv_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstDivShift\(3MLIB\)](#),
[mllib_ImageConstDivShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstDivShift – division into a constant, with shifting

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConstDivShift(mllib_image *dst, const mllib_image *src,  
                                     const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageConstDivShift()` function divides each pixel in an image into a constant value on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] / src[x][y][i] * 2^{shift}$$

In the case of $src[x][y][i] = 0$,

```
dst[x][y][i] = 0           if c[i] = 0  
dst[x][y][i] = DATA_TYPE_MAX if c[i] > 0  
dst[x][y][i] = DATA_TYPE_MIN if c[i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Constant into which each pixel is divided. `c[i]` contains the constant for channel `i`.

shift Left shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstDiv\(3MLIB\)](#), [mllib_ImageConstDiv_Fp\(3MLIB\)](#),
[mllib_ImageConstDiv_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstDiv_Inp\(3MLIB\)](#),
[mllib_ImageConstDivShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstDivShift_Inp – division into a constant, with shifting

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConstDivShift_Inp(mllib_image *srcdst,
      const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageConstDivShift_Inp()` function divides each pixel in an image into a constant value on a pixel-by-pixel basis, in place. It scales the result by a left shift and writes the result to the image on a pixel-by-pixel basis.

It uses the following equation:

$$srcdst[x][y][i] = c[i] / srcdst[x][y][i] * 2^{shift}$$

In the case of $srcdst[x][y][i] = 0$,

```
srcdst[x][y][i] = 0           if c[i] = 0
srcdst[x][y][i] = DATA_TYPE_MAX if c[i] > 0
srcdst[x][y][i] = DATA_TYPE_MIN if c[i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* Constant into which each pixel is divided. `c[i]` contains the constant for channel *i*.
- shift* Left shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstDiv\(3MLIB\)](#), [mllib_ImageConstDiv_Fp\(3MLIB\)](#), [mllib_ImageConstDiv_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstDiv_Inp\(3MLIB\)](#), [mllib_ImageConstDivShift\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstMul – multiplication with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstMul(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *c);
```

Description The `mllib_ImageConstMul()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = c[i] * \text{src}[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Constant by which each pixel is multiplied. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstMul_Fp\(3MLIB\)](#), [mllib_ImageConstMul_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstMul_Inp\(3MLIB\)](#), [mllib_ImageConstMulShift\(3MLIB\)](#), [mllib_ImageConstMulShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstMul_Fp – multiply with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstMul_Fp(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *c);
```

Description The `mllib_ImageConstMul_Fp()` function multiplies each pixel in a floating-point image by a constant value on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] * src[x][y][i]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Constant by which each pixel is multiplied. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstMul\(3MLIB\)](#), [mllib_ImageConstMul_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstMul_Inp\(3MLIB\)](#), [mllib_ImageConstMulShift\(3MLIB\)](#), [mllib_ImageConstMulShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstMul_Fp_Inp – multiply with a constant

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_ImageConstMul_Fp_Inp(mllib_image *srcdst,  
    const mllib_d64 *c);
```

Description The mllib_ImageConstMul_Fp_Inp() function multiplies each pixel in a floating-point image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] * \text{srcdst}[x][y][i]$$

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* Constant by which each pixel is multiplied. *c*[*i*] contains the constant for channel *i*.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstMul\(3MLIB\)](#), [mllib_ImageConstMul_Fp\(3MLIB\)](#),
[mllib_ImageConstMul_Inp\(3MLIB\)](#), [mllib_ImageConstMulShift\(3MLIB\)](#),
[mllib_ImageConstMulShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageConstMul_Inp – multiply with a constant

Synopsis `cc [flag...] file... -lmlib [library...]
#include <mlib.h>`

```
mlib_status mlib_ImageConstMul_Inp(mlib_image *srcdst, const mlib_d64 *c);
```

Description The `mlib_ImageConstMul_Inp()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] * \text{srcdst}[x][y][i]$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

c Constant by which each pixel is multiplied. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageConstMul\(3MLIB\)](#), [mlib_ImageConstMul_Fp\(3MLIB\)](#),
[mlib_ImageConstMul_Fp_Inp\(3MLIB\)](#), [mlib_ImageConstMulShift\(3MLIB\)](#),
[mlib_ImageConstMulShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstMulShift – multiply with a constant, with shifting

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConstMulShift(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageConstMulShift()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] * src[x][y][i] * 2^{*(-shift)}$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Constant by which each pixel is multiplied. `c[i]` contains the constant for channel *i*.
- shift* Right shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstMul\(3MLIB\)](#), [mllib_ImageConstMul_Fp\(3MLIB\)](#), [mllib_ImageConstMul_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstMul_Inp\(3MLIB\)](#), [mllib_ImageConstMulShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstMulShift_Inp – multiply with a constant, with shifting

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConstMulShift_Inp(mllib_image *srcdst,
      const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageConstMulShift_Inp()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] * \text{srcdst}[x][y][i] * 2^{*(-\text{shift})}$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

c Constant by which each pixel is multiplied. `c[i]` contains the constant for channel *i*.

shift Right shifting factor. $0 \leq \text{shift} \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConstMul(3MLIB)`, `mllib_ImageConstMul_Fp(3MLIB)`,
`mllib_ImageConstMul_Fp_Inp(3MLIB)`, `mllib_ImageConstMul_Inp(3MLIB)`,
`mllib_ImageConstMulShift(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConstNotAnd – NotAnd with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstNotAnd(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstNotAnd()` function computes the logical And of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \sim(c[i] \& src[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotAnd_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstNotAnd_Inp – NotAnd with a constant, in place

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstNotAnd_Inp(mllib_image *srcdst,  
                                         const mllib_s32 *c);
```

Description The `mllib_ImageConstNotAnd_Inp()` function computes the logical And of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \sim(c[i] \ \& \ \text{srcdst}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. *c*[*i*] contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotAnd\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstNotOr – NotOr with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstNotOr(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstNotOr()` function computes the logical Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \sim(c[i] \mid src[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotOr_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstNotOr_Inp – NotOr with a constant, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstNotOr_Inp(mllib_image *srcdst,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstNotOr_Inp()` function computes the logical Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$srcdst[x][y][i] = \sim(c[i] \mid srcdst[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotOr\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstNotXor – NotXor with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConstNotXor(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstNotXor()` function computes the logical exclusive Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \sim(c[i] \wedge src[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotXor_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstNotXor_Inp – NotXor with a constant, in place

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConstNotXor_Inp(mllib_image *srcdst,  
    const mllib_s32 *c);
```

Description The `mllib_ImageConstNotXor_Inp()` function computes the logical exclusive Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \sim(c[i] \wedge \text{srcdst}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstNotXor\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstOr – Or with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstOr(mllib_image *dst, const mllib_image *src,  
                                const mllib_s32 *c);
```

Description The `mllib_ImageConstOr()` function computes the logical Or of the source image with a constant on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] \mid src[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstOr_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name `mlib_ImageConstOr_Inp` – Or with a constant, in place

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageConstOr_Inp(mlib_image *srcdst, const mlib_s32 *c);
```

Description The `mlib_ImageConstOr_Inp()` function computes the logical Or of the source image with a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] \mid \text{srcdst}[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

srcdst Pointer to first source and destination image.

c Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageConstOr\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstOrNot – OrNot with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstOrNot(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstOrNot()` function computes the logical Not of the source image and then takes the logical Or of that result with a constant on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] \mid (\sim src[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstOrNot_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstOrNot_Inp – OrNot with a constant, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstOrNot_Inp(mllib_image *srcdst,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstOrNot_Inp()` function computes the logical Not of the source image and then takes the logical Or of that result with a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

$$srcdst[x][y][i] = c[i] \mid (\sim srcdst[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstOrNot\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstSub – Subtraction with a constant

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageConstSub(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *c);
```

Description The `mllib_ImageConstSub()` function subtracts an image pixel from a constant on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = c[i] - \text{src}[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstSub_Fp\(3MLIB\)](#), [mllib_ImageConstSub_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstSub_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstSub_Fp – Subtraction with a constant

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConstSub_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *c);
```

Description The `mllib_ImageConstSub_Fp()` function subtracts a floating-point image pixel from a constant on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] - src[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstSub\(3MLIB\)](#), [mllib_ImageConstSub_Fp_Inp\(3MLIB\)](#), [mllib_ImageConstSub_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstSub_Fp_Inp – subtraction with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstSub_Fp_Inp(mllib_image *srcdst,  
    const mllib_d64 *c);
```

Description The `mllib_ImageConstSub_Fp_Inp()` function subtracts a floating-point image pixel from a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = c[i] - \text{srcdst}[x][y][i]$$

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstSub\(3MLIB\)](#), [mllib_ImageConstSub_Fp\(3MLIB\)](#),
[mllib_ImageConstSub_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstSub_Inp – Subtraction with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageConstSub_Inp(mllib_image *srcdst, const mllib_s32 *c);`

Description The `mllib_ImageConstSub_Inp()` function subtracts an image pixel from a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

$$srcdst[x][y][i] = c[i] - srcdst[x][y][i]$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

c Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstSub\(3MLIB\)](#), [mllib_ImageConstSub_Fp\(3MLIB\)](#),
[mllib_ImageConstSub_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstXor – Xor with a constant

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageConstXor(mllib_image *dst, const mllib_image *src,  
                                const mllib_s32 *c);
```

Description The `mllib_ImageConstXor()` function computes the exclusive Or of the source image with a constant on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = c[i] \wedge src[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

c Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstXor_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConstXor_Inp – Xor with a constant, in place

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageConstXor_Inp(mllib_image *srcdst, const mllib_s32 *c);`

Description The `mllib_ImageConstXor_Inp()` function computes the exclusive Or of the source image with a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

$$srcdst[x][y][i] = c[i] \wedge srcdst[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

- srcdst* Pointer to first source and destination image.
- c* Array of constants to be applied to each pixel. `c[i]` contains the constant for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConstXor\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv2x2 – 2x2 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv2x2(mllib_image *dst, const mllib_image *src,  
    const mllib_s32 *kernel, mllib_s32 scale, mllib_s32 cmask,  
    mllib_edge edge);
```

Description The `mllib_ImageConv2x2()` function performs a 2x2 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (0, 0) of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q] * 2^{**(-scale)}$$

where $m = 2$, $n = 2$, $dm = (m - 1)/2 = 0$, $dn = (n - 1)/2 = 0$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2_Fp(3MLIB)`, `mlib_ImageConv2x2Index(3MLIB)`,
`mlib_ImageConv3x3(3MLIB)`, `mlib_ImageConv3x3_Fp(3MLIB)`,
`mlib_ImageConv3x3Index(3MLIB)`, `mlib_ImageConv4x4(3MLIB)`,
`mlib_ImageConv4x4_Fp(3MLIB)`, `mlib_ImageConv4x4Index(3MLIB)`,
`mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5_Fp(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConv2x2_Fp – 2x2 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv2x2_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConv2x2_Fp()` function performs a 2x2 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at $(0, 0)$ of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q]$$

where $m = 2$, $n = 2$, $dm = (m - 1)/2 = 0$, $dn = (n - 1)/2 = 0$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2Index(3MLIB)`,
`mlib_ImageConv3x3(3MLIB)`, `mlib_ImageConv3x3_Fp(3MLIB)`,
`mlib_ImageConv3x3Index(3MLIB)`, `mlib_ImageConv4x4(3MLIB)`,
`mlib_ImageConv4x4_Fp(3MLIB)`, `mlib_ImageConv4x4Index(3MLIB)`,
`mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5_Fp(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

Name mllib_ImageConv2x2Index – 2x2 convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv2x2Index(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageConv2x2Index()` function performs a 2x2 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at (0, 0) of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single-channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**}(-scale)$$

where $m = 2$, $n = 2$, $dm = (m - 1)/2 = 0$, $dn = (n - 1)/2 = 0$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv3x3\(3MLIB\)](#), [mllib_ImageConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageConv3x3Index\(3MLIB\)](#), [mllib_ImageConv4x4\(3MLIB\)](#),
[mllib_ImageConv4x4_Fp\(3MLIB\)](#), [mllib_ImageConv4x4Index\(3MLIB\)](#),
[mllib_ImageConv5x5\(3MLIB\)](#), [mllib_ImageConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageConv5x5Index\(3MLIB\)](#), [mllib_ImageConv7x7\(3MLIB\)](#),
[mllib_ImageConv7x7_Fp\(3MLIB\)](#), [mllib_ImageConv7x7Index\(3MLIB\)](#),
[mllib_ImageConvKernelConvert\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv3x3 – 3x3 convolution

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv3x3(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_s32 cmask,
    mllib_edge edge);
```

Description The `mllib_ImageConv3x3()` function performs a 3x3 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

The data type of source and destination images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**}(-scale)$$

where $m = 3$, $n = 3$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageConv2x2\(3MLIB\)](#), [mlib_ImageConv2x2_Fp\(3MLIB\)](#),
[mlib_ImageConv2x2Index\(3MLIB\)](#), [mlib_ImageConv3x3_Fp\(3MLIB\)](#),
[mlib_ImageConv3x3Index\(3MLIB\)](#), [mlib_ImageConv4x4\(3MLIB\)](#),
[mlib_ImageConv4x4_Fp\(3MLIB\)](#), [mlib_ImageConv4x4Index\(3MLIB\)](#),
[mlib_ImageConv5x5\(3MLIB\)](#), [mlib_ImageConv5x5_Fp\(3MLIB\)](#),
[mlib_ImageConv5x5Index\(3MLIB\)](#), [mlib_ImageConv7x7\(3MLIB\)](#),
[mlib_ImageConv7x7_Fp\(3MLIB\)](#), [mlib_ImageConv7x7Index\(3MLIB\)](#),
[mlib_ImageConvKernelConvert\(3MLIB\)](#), [mlib_ImageConvMxN\(3MLIB\)](#),
[mlib_ImageConvMxN_Fp\(3MLIB\)](#), [mlib_ImageConvMxNIndex\(3MLIB\)](#),
[mlib_ImageConvolveMxN\(3MLIB\)](#), [mlib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mlib_ImageSConv3x3\(3MLIB\)](#), [mlib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mlib_ImageSConv5x5\(3MLIB\)](#), [mlib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mlib_ImageSConv7x7\(3MLIB\)](#), [mlib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mlib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv3x3_Fp – 3x3 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv3x3_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConv3x3_Fp()` function performs a floating-point 3x3 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q]$$

where $m = 3$, $n = 3$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`,
`mlib_ImageConv2x2Index(3MLIB)`, `mlib_ImageConv3x3(3MLIB)`,
`mlib_ImageConv3x3Index(3MLIB)`, `mlib_ImageConv4x4(3MLIB)`,
`mlib_ImageConv4x4_Fp(3MLIB)`, `mlib_ImageConv4x4Index(3MLIB)`,
`mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5_Fp(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

Name mllib_ImageConv3x3Index – 3x3 convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv3x3Index(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageConv3x3Index()` function performs a 3x3 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single-channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**}(-scale)$$

where $m = 3$, $n = 3$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function. The source and destination images must be single-channel images.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv4x4\(3MLIB\)](#),
[mllib_ImageConv4x4_Fp\(3MLIB\)](#), [mllib_ImageConv4x4Index\(3MLIB\)](#),
[mllib_ImageConv5x5\(3MLIB\)](#), [mllib_ImageConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageConv5x5Index\(3MLIB\)](#), [mllib_ImageConv7x7\(3MLIB\)](#),
[mllib_ImageConv7x7_Fp\(3MLIB\)](#), [mllib_ImageConv7x7Index\(3MLIB\)](#),
[mllib_ImageConvKernelConvert\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv4x4 – 4x4 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv4x4(mllib_image *dst, const mllib_image *src,  
                                const mllib_s32 *kernel, mllib_s32 scale, mllib_s32 cmask,  
                                mllib_edge edge);
```

Description The `mllib_ImageConv4x4()` function performs a 4x4 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (1, 1) of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q] * 2^{**(-scale)}$$

where $m = 4$, $n = 4$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```


Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`,
`mlib_ImageConv2x2Index(3MLIB)`, `mlib_ImageConv3x3(3MLIB)`,
`mlib_ImageConv3x3_Fp(3MLIB)`, `mlib_ImageConv3x3Index(3MLIB)`,
`mlib_ImageConv4x4_Fp(3MLIB)`, `mlib_ImageConv4x4Index(3MLIB)`,
`mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5_Fp(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConv4x4_Fp – 4x4 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv4x4_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConv4x4_Fp()` function performs a floating-point 4x4 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (1, 1) of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q]$$

where $m = 4$, $n = 4$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`,
`mlib_ImageConv2x2Index(3MLIB)`, `mlib_ImageConv3x3(3MLIB)`,
`mlib_ImageConv3x3_Fp(3MLIB)`, `mlib_ImageConv3x3Index(3MLIB)`,
`mlib_ImageConv4x4(3MLIB)`, `mlib_ImageConv4x4Index(3MLIB)`,
`mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5_Fp(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

Name mllib_ImageConv4x4Index – 4x4 convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv4x4Index(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageConv4x4Index()` function performs a 4x4 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at (1, 1) of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**}(-scale)$$

where $m = 4$, $n = 4$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv5x5\(3MLIB\)](#), [mllib_ImageConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageConv5x5Index\(3MLIB\)](#), [mllib_ImageConv7x7\(3MLIB\)](#),
[mllib_ImageConv7x7_Fp\(3MLIB\)](#), [mllib_ImageConv7x7Index\(3MLIB\)](#),
[mllib_ImageConvKernelConvert\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv5x5 – 5x5 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv5x5(mllib_image *dst, const mllib_image *src,  
    const mllib_s32 *kernel, mllib_s32 scale, mllib_s32 cmask,  
    mllib_edge edge);
```

Description The `mllib_ImageConv5x5()` function performs a 5x5 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q] * 2^{**(-scale)}$$

where $m = 5$, $n = 5$, $dm = (m - 1)/2 = 2$, $dn = (n - 1)/2 = 2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5_Fp(3MLIB)`,
`mllib_ImageConv5x5Index(3MLIB)`, `mllib_ImageConv7x7(3MLIB)`,
`mllib_ImageConv7x7_Fp(3MLIB)`, `mllib_ImageConv7x7Index(3MLIB)`,
`mllib_ImageConvKernelConvert(3MLIB)`, `mllib_ImageConvMxN(3MLIB)`,
`mllib_ImageConvMxN_Fp(3MLIB)`, `mllib_ImageConvMxNIndex(3MLIB)`,
`mllib_ImageConvolveMxN(3MLIB)`, `mllib_ImageConvolveMxN_Fp(3MLIB)`,
`mllib_ImageSConv3x3(3MLIB)`, `mllib_ImageSConv3x3_Fp(3MLIB)`,
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,
`mllib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConv5x5_Fp – 5x5 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv5x5_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConv5x5_Fp()` function performs a floating-point 5x5 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q]$$

where $m = 5$, $n = 5$, $dm = (m - 1)/2 = 2$, $dn = (n - 1)/2 = 2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`,
`mlib_ImageConv2x2Index(3MLIB)`, `mlib_ImageConv3x3(3MLIB)`,
`mlib_ImageConv3x3_Fp(3MLIB)`, `mlib_ImageConv3x3Index(3MLIB)`,
`mlib_ImageConv4x4(3MLIB)`, `mlib_ImageConv4x4_Fp(3MLIB)`,
`mlib_ImageConv4x4Index(3MLIB)`, `mlib_ImageConv5x5(3MLIB)`,
`mlib_ImageConv5x5Index(3MLIB)`, `mlib_ImageConv7x7(3MLIB)`,
`mlib_ImageConv7x7_Fp(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

Name mllib_ImageConv5x5Index – 5x5 convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv5x5Index(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageConv5x5Index()` function performs a 5x5 convolution the color indexed source image by using the user-supplied kernel. The source and destination images must be single-channel images.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{*(-scale)}$$

where $m = 5$, $n = 5$, $dm = (m - 1)/2 = 2$, $dn = (n - 1)/2 = 2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#),
[mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv7x7\(3MLIB\)](#),
[mllib_ImageConv7x7_Fp\(3MLIB\)](#), [mllib_ImageConv7x7Index\(3MLIB\)](#),
[mllib_ImageConvKernelConvert\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConv7x7 – 7x7 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv7x7(mllib_image *dst, const mllib_image *src,  
    const mllib_s32 *kernel, mllib_s32 scale, mllib_s32 cmask,  
    mllib_edge edge);
```

Description The `mllib_ImageConv7x7()` function performs a 7x7 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q] * 2^{**(-scale)}$$

where $m = 7$, $n = 7$, $dm = (m - 1)/2 = 3$, $dn = (n - 1)/2 = 3$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,
`mllib_ImageConv7x7_Fp(3MLIB)`, `mllib_ImageConv7x7Index(3MLIB)`,
`mllib_ImageConvKernelConvert(3MLIB)`, `mllib_ImageConvMxN(3MLIB)`,
`mllib_ImageConvMxN_Fp(3MLIB)`, `mllib_ImageConvMxNIndex(3MLIB)`,
`mllib_ImageConvolveMxN(3MLIB)`, `mllib_ImageConvolveMxN_Fp(3MLIB)`,
`mllib_ImageSConv3x3(3MLIB)`, `mllib_ImageSConv3x3_Fp(3MLIB)`,
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,
`mllib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConv7x7_Fp – 7x7 convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConv7x7_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConv7x7_Fp()` function performs a floating-point 7x7 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q]$$

where $m = 7$, $n = 7$, $dm = (m - 1)/2 = 3$, $dn = (n - 1)/2 = 3$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC  
MLIB_EDGE_SRC_EXTEND
```

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`,
`mlib_ImageConv2x2Index(3MLIB)`, `mlib_ImageConv3x3(3MLIB)`,
`mlib_ImageConv3x3_Fp(3MLIB)`, `mlib_ImageConv3x3Index(3MLIB)`,
`mlib_ImageConv4x4(3MLIB)`, `mlib_ImageConv4x4_Fp(3MLIB)`,
`mlib_ImageConv4x4Index(3MLIB)`, `mlib_ImageConv5x5(3MLIB)`,
`mlib_ImageConv5x5_Fp(3MLIB)`, `mlib_ImageConv5x5Index(3MLIB)`,
`mlib_ImageConv7x7(3MLIB)`, `mlib_ImageConv7x7Index(3MLIB)`,
`mlib_ImageConvKernelConvert(3MLIB)`, `mlib_ImageConvMxN(3MLIB)`,
`mlib_ImageConvMxN_Fp(3MLIB)`, `mlib_ImageConvMxNIndex(3MLIB)`,
`mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`,
`mlib_ImageSConv3x3(3MLIB)`, `mlib_ImageSConv3x3_Fp(3MLIB)`,
`mlib_ImageSConv5x5(3MLIB)`, `mlib_ImageSConv5x5_Fp(3MLIB)`,
`mlib_ImageSConv7x7(3MLIB)`, `mlib_ImageSConv7x7_Fp(3MLIB)`,
`mlib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

Name mllib_ImageConv7x7Index – 7x7 convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConv7x7Index(mllib_image *dst, const mllib_image *src,
    const mllib_s32 *kernel, mllib_s32 scale, mllib_edge edge,
    const void *colormap);
```

Description The `mllib_ImageConv7x7Index()` function performs a 7x7 convolution the color indexed source image by using the user-supplied kernel. The source and destination images must be single-channel images.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**(-scale)}$$

where $m = 7$, $n = 7$, $dm = (m - 1)/2 = 3$, $dn = (n - 1)/2 = 3$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

scale Scaling factor.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#), [mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#), [mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#), [mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#), [mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#), [mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv5x5Index\(3MLIB\)](#), [mllib_ImageConv7x7\(3MLIB\)](#), [mllib_ImageConv7x7_Fp\(3MLIB\)](#), [mllib_ImageConvKernelConvert\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#), [mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#), [mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#), [mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#), [mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#), [mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#), [mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConvKernelConvert – convolution kernel conversion

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConvKernelConvert(mllib_s32 *ikernel, mllib_s32 *iscale,  
    const mllib_d64 *fkernel, mllib_s32 m, mllib_s32 n, mllib_type type);
```

Description The `mllib_ImageConvKernelConvert()` function converts a floating-point convolution kernel to an integer kernel with its scaling factor suitable to be used in convolution functions.

Parameters The function takes the following arguments:

- ikernel* Pointer to integer convolution kernel, in row major order.
- iscale* Pointer to scaling factor of the integer convolution kernel.
- fkernel* Pointer to floating-point convolution kernel, in row major order.
- m* Width of the convolution kernel. $m \geq 1$.
- n* Height of the convolution kernel. $n \geq 1$.
- type* The image type. It can be one of the following:
 - MLIB_BIT
 - MLIB_BYTE
 - MLIB_SHORT
 - MLIB_USHORT
 - MLIB_INT

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#),
[mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv5x5Index\(3MLIB\)](#),
[mllib_ImageConv7x7\(3MLIB\)](#), [mllib_ImageConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageConv7x7Index\(3MLIB\)](#), [mllib_ImageConvMxN\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),

```
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),  
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),  
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),  
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
```

Name mllib_ImageConvMxN – MxN convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConvMxN(mllib_image *dst, const mllib_image *src,  
                                const mlib_s32 *kernel, mlib_s32 m, mlib_s32 n, mlib_s32 dm,  
                                mlib_s32 dn, mlib_s32 scale, mlib_s32 cmask, mlib_edge edge);
```

Description The `mllib_ImageConvMxN()` function performs a MxN convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (dm, dn) of the kernel matrix.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{**}(-scale)$$

where $m \geq 1$, $n \geq 1$, $0 \leq dm < m$, $0 \leq dn < n$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

m Width of the convolution kernel. $m \geq 1$.

n Height of the convolution kernel. $n \geq 1$.

dm X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.

dn Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.

scale Scaling factor.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#),
[mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv5x5Index\(3MLIB\)](#),
[mllib_ImageConv7x7\(3MLIB\)](#), [mllib_ImageConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageConv7x7Index\(3MLIB\)](#), [mllib_ImageConvKernelConvert\(3MLIB\)](#),
[mllib_ImageConvMxN_Fp\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConvMxN_Fp – MxN convolution

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConvMxN_Fp(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *kernel, mllib_s32 m, mllib_s32 n, mllib_s32 dm,  
    mllib_s32 dn, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConvMxN_Fp()` function performs a MxN convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (dm, dn) of the kernel matrix.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q]$$

where $m \geq 1$, $n \geq 1$, $0 \leq dm < m$, $0 \leq dn < n$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

kernel Pointer to the convolution kernel, in row major order.

m Width of the convolution kernel. $m \geq 1$.

n Height of the convolution kernel. $n \geq 1$.

dm X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.

dn Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.

cmask Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

edge Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE  
MLIB_EDGE_DST_FILL_ZERO  
MLIB_EDGE_DST_COPY_SRC
```

MLIB_EDGE_SRC_EXTEND

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#),
[mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv5x5Index\(3MLIB\)](#),
[mllib_ImageConv7x7\(3MLIB\)](#), [mllib_ImageConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageConv7x7Index\(3MLIB\)](#), [mllib_ImageConvKernelConvert\(3MLIB\)](#),
[mllib_ImageConvMxN\(3MLIB\)](#), [mllib_ImageConvMxNIndex\(3MLIB\)](#),
[mllib_ImageConvolveMxN\(3MLIB\)](#), [mllib_ImageConvolveMxN_Fp\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageConvMxNIndex – MxN convolution on a color indexed image

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageConvMxNIndex(mllib_image *dst, const mllib_image *src,
                                     const mllib_s32 *kernel, mllib_s32 m, mllib_s32 n, mllib_s32 dm,
                                     mllib_s32 dn, mllib_s32 scale, mllib_edge edge, const void *colormap);
```

Description The `mllib_ImageConvMxNIndex()` function performs a MxN convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at (dm, dn) of the kernel matrix.

This function performs the convolution on a color indexed image. The input image and the output image must be single-channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

$$\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] * k[p][q] * 2^{*(-scale)}$$

where $m > 1$, $n > 1$, $0 \leq dm < m$, $0 \leq dn < n$.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>kernel</i>	Pointer to the convolution kernel, in row major order.
<i>m</i>	Width of the convolution kernel. $m > 1$.
<i>n</i>	Height of the convolution kernel. $n > 1$.
<i>dm</i>	X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
<i>scale</i>	Scaling factor.
<i>edge</i>	Type of edge condition. It can be one of the following: <code>MLIB_EDGE_DST_NO_WRITE</code> <code>MLIB_EDGE_DST_FILL_ZERO</code> <code>MLIB_EDGE_DST_COPY_SRC</code> <code>MLIB_EDGE_SRC_EXTEND</code>

colormap Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,
`mllib_ImageConvolveMxN(3MLIB)`, `mllib_ImageConvolveMxN_Fp(3MLIB)`,
`mllib_ImageSConv3x3(3MLIB)`, `mllib_ImageSConv3x3_Fp(3MLIB)`,
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,
`mllib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConvolveMxN – MxN convolution, with kernel analysis for taking advantage of special cases

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConvolveMxN(mllib_image *dst, const mllib_image *src,
    const mllib_d64 *kernel, mllib_s32 m, mllib_s32 n, mllib_s32 dm,
    mllib_s32 dn, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConvolveMxN()` function analyzes the convolution kernel, converts the floating-point kernel to an integer kernel, then performs a MxN convolution on the source image by calling either one of the functions like `mllib_ImageSConv3x3()`, `mllib_ImageConv3x3()`, and etc. in special cases or `mllib_ImageConvMxN()` in other cases.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q]$$

where $m \geq 1$, $n \geq 1$, $0 \leq dm < m$, $0 \leq dn < n$.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>kernel</i>	Pointer to the convolution kernel, in row major order.
<i>m</i>	Width of the convolution kernel. $m \geq 1$.
<i>n</i>	Height of the convolution kernel. $n \geq 1$.
<i>dm</i>	X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
<i>cmask</i>	Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_DST_FILL_ZERO MLIB_EDGE_DST_COPY_SRC MLIB_EDGE_SRC_EXTEND

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN_Fp(3MLIB)`,
`mllib_ImageSConv3x3(3MLIB)`, `mllib_ImageSConv3x3_Fp(3MLIB)`,
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,
`mllib_ImageSConvKernelConvert(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageConvolveMxN_Fp – MxN convolution, with kernel analysis for taking advantage of special cases

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageConvolveMxN_Fp(mllib_image *dst,  
    const mllib_image *src, const mllib_d64 *kernel, mllib_s32 m,  
    mllib_s32 n, mllib_s32 dm, mllib_s32 dn, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageConvolveMxN_Fp()` function analyzes the convolution kernel, then performs a MxN convolution on the source image by calling either one of the functions like `mllib_ImageSConv3x3_Fp()`, `mllib_ImageConv3x3_Fp()`, and etc. in special cases or `mllib_ImageConvMxN_Fp()` in other cases.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q]$$

where $m \geq 1$, $n \geq 1$, $0 \leq dm < m$, $0 \leq dn < n$.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>kernel</i>	Pointer to the convolution kernel, in row major order.
<i>m</i>	Width of the convolution kernel. $m \geq 1$.
<i>n</i>	Height of the convolution kernel. $n \geq 1$.
<i>dm</i>	X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
<i>cmask</i>	Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_DST_FILL_ZERO MLIB_EDGE_DST_COPY_SRC MLIB_EDGE_SRC_EXTEND

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageConv2x2\(3MLIB\)](#), [mllib_ImageConv2x2_Fp\(3MLIB\)](#),
[mllib_ImageConv2x2Index\(3MLIB\)](#), [mllib_ImageConv3x3\(3MLIB\)](#),
[mllib_ImageConv3x3_Fp\(3MLIB\)](#), [mllib_ImageConv3x3Index\(3MLIB\)](#),
[mllib_ImageConv4x4\(3MLIB\)](#), [mllib_ImageConv4x4_Fp\(3MLIB\)](#),
[mllib_ImageConv4x4Index\(3MLIB\)](#), [mllib_ImageConv5x5\(3MLIB\)](#),
[mllib_ImageConv5x5_Fp\(3MLIB\)](#), [mllib_ImageConv5x5Index\(3MLIB\)](#),
[mllib_ImageConv7x7\(3MLIB\)](#), [mllib_ImageConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageConv7x7Index\(3MLIB\)](#), [mllib_ImageConvKernelConvert\(3MLIB\)](#),
[mllib_ImageConvMxN\(3MLIB\)](#), [mllib_ImageConvMxN_Fp\(3MLIB\)](#),
[mllib_ImageConvMxNIndex\(3MLIB\)](#), [mllib_ImageConvolveMxN\(3MLIB\)](#),
[mllib_ImageSConv3x3\(3MLIB\)](#), [mllib_ImageSConv3x3_Fp\(3MLIB\)](#),
[mllib_ImageSConv5x5\(3MLIB\)](#), [mllib_ImageSConv5x5_Fp\(3MLIB\)](#),
[mllib_ImageSConv7x7\(3MLIB\)](#), [mllib_ImageSConv7x7_Fp\(3MLIB\)](#),
[mllib_ImageSConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCopy – image copy

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageCopy(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageCopy()` function copies the source image to the destination image. The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

It uses the following equation:

`dst[x][y][i] = src[x][y][i]`

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCopyArea\(3MLIB\)](#), [mllib_ImageCopyMask\(3MLIB\)](#),
[mllib_ImageCopyMask_Fp\(3MLIB\)](#), [mllib_ImageCopySubimage\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCopyArea – copy an area

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageCopyArea(mllib_image *img, mllib_s32 x, mllib_s32 y,
                                mllib_s32 w, mllib_s32 h, mllib_s32 dx, mllib_s32 dy);
```

Description The `mllib_ImageCopyArea()` function copies a specified rectangular area from one portion of the image to another portion of the same image. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

It uses the following equation:

$$\text{img}[x+dx+i][y+dy+j][i] = \text{img}[x+i][y+j][i]$$

where $i = 0, 1, \dots, w-1$; $j = 0, 1, \dots, h-1$.

Parameters The function takes the following arguments:

- img* Pointer to source image.
- x* X coordinate of the area origin in the source.
- y* Y coordinate of the area origin in the source.
- w* Width of the area to be copied.
- h* Height of the area to be copied.
- dx* Horizontal displacement in pixels of the area to be copied.
- dy* Vertical displacement in pixels of the area to be copied.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCopy\(3MLIB\)](#), [mllib_ImageCopyMask\(3MLIB\)](#), [mllib_ImageCopyMask_Fp\(3MLIB\)](#), [mllib_ImageCopySubimage\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCopyMask – copy with mask

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageCopyMask(mllib_image *dst, const mllib_image *src,  
                                const mllib_image *mask, const mllib_s32 *thresh);
```

Description The `mllib_ImageCopyMask()` function copies one image to another image via a mask image by using it as a yes/no indicator. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i]  if mask[x][y][i] ≤ thresh[i]  
dst[x][y][i] = dst[x][y][i]  if mask[x][y][i] > thresh[i]
```

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mask* Pointer to mask image.
- thresh* Threshold for the mask image. `thresh[i]` contains the threshold for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCopy\(3MLIB\)](#), [mllib_ImageCopyArea\(3MLIB\)](#), [mllib_ImageCopyMask_Fp\(3MLIB\)](#), [mllib_ImageCopySubimage\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCopyMask_Fp – copy with mask, floating-point

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageCopyMask_Fp(mllib_image *dst, const mllib_image *src,
    const mllib_image *mask, const mllib_d64 *thresh);
```

Description The `mllib_ImageCopyMask_Fp()` function copies one image to another image via a mask image by using it as a yes/no indicator. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i] if mask[x][y][i] ≤ thresh[i]
dst[x][y][i] = dst[x][y][i] if mask[x][y][i] > thresh[i]
```

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

mask Pointer to mask image.

thresh Threshold for the mask image. `thresh[i]` contains the threshold for channel *i*.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCopy\(3MLIB\)](#), [mllib_ImageCopyArea\(3MLIB\)](#), [mllib_ImageCopyMask\(3MLIB\)](#), [mllib_ImageCopySubimage\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCopySubimage – copy subimage

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageCopySubimage(mllib_image *dst, const mllib_image *src,  
                                     mllib_s32 xd, mllib_s32 yd, mllib_s32 xs, mllib_s32 ys, mllib_s32 w,  
                                     mllib_s32 h);
```

Description The `mllib_ImageCopySubimage()` function copies a specified rectangular area from one image to a specified area of another image. The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

It uses the following equation:

$$dst[xd+i][yd+j][i] = src[xs+i][ys+j][i]$$

where $i = 0, 1, \dots, w-1$; $j = 0, 1, \dots, h-1$.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- xd* X coordinate of the area origin in the destination.
- yd* Y coordinate of the area origin in the destination.
- xs* X coordinate of the area origin in the source.
- ys* Y coordinate of the area origin in the source.
- w* Width of the area to be copied.
- h* Height of the area to be copied.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCopy\(3MLIB\)](#), [mllib_ImageCopyArea\(3MLIB\)](#), [mllib_ImageCopyMask\(3MLIB\)](#), [mllib_ImageCopyMask_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageCreate – image creation

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```

```
mlib_image *mlib_ImageCreate(mlib_type type, mlib_s32 channels,
                             mlib_s32 width, mlib_s32 height);
```

Description The `mlib_ImageCreate()` function creates a mediaLib image data structure and allocates memory space for image data. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

To obtain the best performance, it is recommended that you use this function to create a mediaLib image whenever possible, as this guarantees alignment.

Parameters The function takes the following arguments:

type Image data type.
channels Number of channels in the image.
width Width of image in pixels.
height Height of image in pixels.

Return Values The function returns a pointer to the `mlib_image` data structure.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageCreateStruct\(3MLIB\)](#), [mlib_ImageCreateSubimage\(3MLIB\)](#), [mlib_ImageDelete\(3MLIB\)](#), [mlib_ImageSetPaddings\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCreateStruct – image structure creation

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_image *mllib_ImageCreateStruct(mllib_type type, mllib_s32 channels,
                                     mllib_s32 width, mllib_s32 height, mllib_s32 stride, const void *datbuf);
```

Description The `mllib_ImageCreateStruct()` function creates a mediaLib image data structure with parameters supplied by the user.

Parameters The function takes the following arguments:

<i>type</i>	Image data type. It can be <code>MLIB_BIT</code> , <code>MLIB_BYTE</code> , <code>MLIB_SHORT</code> , <code>MLIB_USHORT</code> , <code>MLIB_INT</code> , <code>MLIB_FLOAT</code> , or <code>MLIB_DOUBLE</code> .
<i>channels</i>	Number of channels in the image.
<i>width</i>	Width of image in pixels.
<i>height</i>	Height of image in pixels.
<i>stride</i>	Stride of each row of the data space in bytes.
<i>datbuf</i>	Pointer to the image data buffer.

Return Values The function returns a pointer to the `mllib_image` data structure.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageCreate\(3MLIB\)](#), [mllib_ImageCreateSubimage\(3MLIB\)](#),
[mllib_ImageSetStruct\(3MLIB\)](#), [mllib_ImageResetStruct\(3MLIB\)](#),
[mllib_ImageDelete\(3MLIB\)](#), [mllib_ImageSetFormat\(3MLIB\)](#),
[mllib_ImageSetPaddings\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCreateSubimage – subimage creation

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_image *mllib_ImageCreateSubimage(mllib_image *img, mllib_s32 x,
    mllib_s32 y, mllib_s32 w, mllib_s32 h);
```

Description The `mllib_ImageCreateSubimage()` function creates a mediaLib image data structure for a subimage based on a source image. Note that the memory space of the source image data is used for the subimage data. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

- img* Pointer to source image.
- x* X coordinate of subimage origin in the source.
- y* Y coordinate of subimage origin in the source.
- w* Width of the subimage in pixels.
- h* Height of the subimage in pixels.

Return Values The function returns a pointer to the `mllib_image` data structure.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageCreate(3MLIB)`, `mllib_ImageCreateStruct(3MLIB)`,
`mllib_ImageDelete(3MLIB)`, `mllib_ImageSetPaddings(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageCrossCorrel – cross correlation

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageCrossCorrel(mllib_d64 *correl, const mllib_image *img1,  
    const mllib_image *img2);
```

Description The `mllib_ImageCrossCorrel()` function computes the cross-correlation between a pair of images.

It uses the following equation:

$$\text{correl}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] * \text{img2}[x][y][i])$$

where *w* and *h* are the width and height of the images, respectively.

Parameters The function takes the following arguments:

- correl* Pointer to cross correlation array on a channel basis. The array must be the size of channels in the images. `correl[i]` contains the cross-correlation of channel *i*.
- img1* Pointer to first image.
- img2* Pointer to second image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAutoCorrel\(3MLIB\)](#), [mllib_ImageAutoCorrel_Fp\(3MLIB\)](#),
[mllib_ImageCrossCorrel_Fp\(3MLIB\)](#), [mllib_ImageNormCrossCorrel\(3MLIB\)](#),
[mllib_ImageNormCrossCorrel_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageCrossCorrel_Fp – cross correlation

Synopsis cc [*flag...*] *file...* -lmllib [*library...*]
#include <mllib.h>

```
mllib_status mllib_ImageCrossCorrel_Fp(mllib_d64 *correl, const mllib_image *img1,  
    const mllib_image *img2);
```

Description The `mllib_ImageCrossCorrel_Fp()` function computes the cross-correlation between a pair of floating-point images.

It uses the following equation:

$$\text{correl}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] * \text{img2}[x][y][i])$$

where *w* and *h* are the width and height of the images, respectively.

Parameters The function takes the following arguments:

- correl* Pointer to cross correlation array on a channel basis. The array must be the size of channels in the images. `correl[i]` contains the cross-correlation of channel *i*.
- img1* Pointer to first image.
- img2* Pointer to second image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageAutoCorrel\(3MLIB\)](#), [mllib_ImageAutoCorrel_Fp\(3MLIB\)](#),
[mllib_ImageCrossCorrel\(3MLIB\)](#), [mllib_ImageNormCrossCorrel\(3MLIB\)](#),
[mllib_ImageNormCrossCorrel_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDataTypeConvert – data type conversion

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageDataTypeConvert(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageDataTypeConvert()` function converts between data types MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, and MLIB_DOUBLE.

The input and output data images must have the same width, height, and number of channels. Conversion to a smaller pixel format clamps the source value to the dynamic range of the destination pixel.

See the following table for available variations of the data type conversion function.

Source Type	Dest. Type	Action
MLIB_BYTE	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_SHORT	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_USHORT	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_INT	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_FLOAT	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_DOUBLE	MLIB_BIT	(x > 0) ? 1 : 0
MLIB_BIT	MLIB_BYTE	(x == 1) ? 1 : 0
MLIB_SHORT	MLIB_BYTE	(mllib_u8)clamp(x, 0, 255)
MLIB_USHORT	MLIB_BYTE	(mllib_u8)clamp(x, 0, 255)
MLIB_INT	MLIB_BYTE	(mllib_u8)clamp(x, 0, 255)
MLIB_FLOAT	MLIB_BYTE	(mllib_u8)clamp(x, 0, 255)
MLIB_DOUBLE	MLIB_BYTE	(mllib_u8)clamp(x, 0, 255)
MLIB_BIT	MLIB_SHORT	(x == 1) ? 1 : 0
MLIB_BYTE	MLIB_SHORT	(mllib_s16)x
MLIB_USHORT	MLIB_SHORT	(mllib_s16)clamp(x, -32768, 32767)
MLIB_INT	MLIB_SHORT	(mllib_s16)clamp(x, -32768, 32767)
MLIB_FLOAT	MLIB_SHORT	(mllib_s16)clamp(x, -32768, 32767)

Source Type	Dest. Type	Action
MLIB_DOUBLE	MLIB_SHORT	(mllib_s16)clamp(x, -32768, 32767)
MLIB_BIT	MLIB_USHORT	(x == 1) ? 1 : 0
MLIB_BYTE	MLIB_USHORT	(mllib_u16)x
MLIB_SHORT	MLIB_USHORT	(mllib_u16)clamp(x, 0, 65535)
MLIB_INT	MLIB_USHORT	(mllib_u16)clamp(x, 0, 65535)
MLIB_FLOAT	MLIB_USHORT	(mllib_u16)clamp(x, 0, 65535)
MLIB_DOUBLE	MLIB_USHORT	(mllib_u16)clamp(x, 0, 65535)
MLIB_BIT	MLIB_INT	(x == 1) ? 1 : 0
MLIB_BYTE	MLIB_INT	(mllib_s32)x
MLIB_SHORT	MLIB_INT	(mllib_s32)x
MLIB_USHORT	MLIB_INT	(mllib_s32)x
MLIB_FLOAT	MLIB_INT	(mllib_s32)clamp(x, -2147483647-1, 2147483647)
MLIB_DOUBLE	MLIB_INT	(mllib_s32)clamp(x, -2147483647-1, 2147483647)
MLIB_BIT	MLIB_FLOAT	(x == 1) ? 1.0 : 0.0
MLIB_BYTE	MLIB_FLOAT	(mllib_f32)x
MLIB_SHORT	MLIB_FLOAT	(mllib_f32)x
MLIB_USHORT	MLIB_FLOAT	(mllib_f32)x
MLIB_INT	MLIB_FLOAT	(mllib_f32)x
MLIB_DOUBLE	MLIB_FLOAT	(mllib_f32)x
MLIB_BIT	MLIB_DOUBLE	(x == 1) ? 1.0 : 0.0
MLIB_BYTE	MLIB_DOUBLE	(mllib_d64)x
MLIB_SHORT	MLIB_DOUBLE	(mllib_d64)x
MLIB_USHORT	MLIB_DOUBLE	(mllib_d64)x
MLIB_INT	MLIB_DOUBLE	(mllib_d64)x
MLIB_FLOAT	MLIB_DOUBLE	(mllib_d64)x

The actions are defined in C-style pseudo-code. All type casts follow the rules of standard C. `clamp()` can be defined as a macro: `#define clamp(x, low, high) (((x) < (low)) ? (low) : (((x) > (high)) ? (high) : (x)))`

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageReformat\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageDelete – image delete

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

`void mlib_ImageDelete(mlib_image *img);`

Description The `mlib_ImageDelete()` function deletes the mediaLib image data structure and frees the memory space of the image data only if it is allocated through `mlib_ImageCreate()`. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

img Pointer to mediaLib image structure.

Return Values None.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageCreate\(3MLIB\)](#), [mlib_ImageCreateStruct\(3MLIB\)](#),
[mlib_ImageCreateSubimage\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDilate4 – four neighbor dilate

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageDilate4(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageDilate4()` function performs a dilation operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 4-neighbor binary images, it uses the following equation:

$$\text{dst}[x][y][0] = \text{OR}\{ \text{src}[x][y][0], \\ \text{src}[x-1][y][0], \text{src}[x+1][y][0], \\ \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}$$

For 4-neighbor grayscale images, it uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[x][y][0], \\ \text{src}[x-1][y][0], \text{src}[x+1][y][0], \\ \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image
src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDilate4_Fp\(3MLIB\)](#), [mllib_ImageDilate8\(3MLIB\)](#),
[mllib_ImageDilate8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDilate4_Fp – four neighbor dilate

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDilate4_Fp(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageDilate4_Fp()` function performs a floating-point dilation operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 4-neighbor grayscale images, it uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[x][y][0], \\ \text{src}[x-1][y][0], \text{src}[x+1][y][0], \\ \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDilate4\(3MLIB\)](#), [mllib_ImageDilate8\(3MLIB\)](#),
[mllib_ImageDilate8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDilate8 – eight neighbor dilate

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageDilate8(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageDilate8()` function performs a dilation operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 8-neighbor binary images, it uses the following equation:

$$dst[x][y][0] = OR\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

For 8-neighbor grayscale images, it uses the following equation:

$$dst[x][y][0] = MAX\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDilate4\(3MLIB\)](#), [mllib_ImageDilate4_Fp\(3MLIB\)](#),
[mllib_ImageDilate8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDilate8_Fp – eight neighbor dilate

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDilate8_Fp(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageDilate8_Fp()` function performs a floating-point dilation operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 8-neighbor grayscale images, it uses the following equation:

$$dst[x][y][0] = \text{MAX}\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDilate4\(3MLIB\)](#), [mllib_ImageDilate4_Fp\(3MLIB\)](#),
[mllib_ImageDilate8\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDiv1_Fp_Inp – division, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageDiv1_Fp_Inp(mllib_image *src1dst,
    const mllib_image *src2);
```

Description The `mllib_ImageDiv1_Fp_Inp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] / \text{src2}[x][y][i]$$

where the operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDiv_Fp\(3MLIB\)](#), [mllib_ImageDiv2_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDiv2_Fp_Inp – division, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDiv2_Fp_Inp(mllib_image *src2dst,
    const mllib_image *src1);
```

Description The `mllib_ImageDiv2_Fp_Inp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src2dst}[x][y][i] = \text{src1}[x][y][i] / \text{src2dst}[x][y][i]$$

where the operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

src2dst Pointer to second source and destination image.

src1 Pointer to first source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDiv_Fp\(3MLIB\)](#), [mllib_ImageDiv1_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivAlpha – alpha channel division

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDivAlpha(mllib_image *dst, const mllib_image *src,
                                mllib_s32 cmask);
```

Description The `mllib_ImageDivAlpha()` function divides color channels by the alpha channel on a pixel-by-pixel basis.

For the `MLIB_BYTE` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] / (\text{src}[x][y][a] * 2^{**(-8)})$$

For the `MLIB_SHORT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] / (\text{src}[x][y][a] * 2^{**(-15)})$$

For the `MLIB_USHORT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] / (\text{src}[x][y][a] * 2^{**(-16)})$$

For the `MLIB_INT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] / (\text{src}[x][y][a] * 2^{**(-31)})$$

where `c` and `a` are the indices for the color channels and the alpha channel, respectively, so `c != a`.

In the case of `src[x][y][a] = 0`,

```
dst[x][y][c] = 0          if src[x][y][c] = 0
dst[x][y][c] = DATA_TYPE_MAX if src[x][y][c] > 0
dst[x][y][c] = DATA_TYPE_MIN if src[x][y][c] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

cmask Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageDivAlpha_Inp\(3MLIB\)](#), [mlib_ImageDivAlpha_Fp\(3MLIB\)](#),
[mlib_ImageDivAlpha_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivAlpha_Fp – alpha channel division

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageDivAlpha_Fp(mllib_image *dst, const mllib_image *src,
    mllib_s32 cmask);
```

Description The `mllib_ImageDivAlpha_Fp()` function divides floating-point color channels by the alpha channel on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][c] = src[x][y][c] / src[x][y][a]$$

where `c` and `a` are the indices for the color channels and the alpha channel, respectively, so `c != a`.

The operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmask* Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivAlpha\(3MLIB\)](#), [mllib_ImageDivAlpha_Fp_Inp\(3MLIB\)](#), [mllib_ImageDivAlpha_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageDivAlpha_Fp_Inp – alpha channel division, in place

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```

```
mlib_status mlib_ImageDivAlpha_Fp_Inp(mlib_image *srcdst, mlib_s32 cmask);
```

Description The `mlib_ImageDivAlpha_Fp_Inp()` function divides floating-point color channels by the alpha channel on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] / \text{srcdst}[x][y][a]$$

where *c* and *a* are the indices for the color channels and the alpha channel, respectively, so *c* != *a*.

The operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

cmask Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of *cmask* is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageDivAlpha\(3MLIB\)](#), [mlib_ImageDivAlpha_Fp\(3MLIB\)](#), [mlib_ImageDivAlpha_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivAlpha_Inp – alpha channel division, in place

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageDivAlpha_Inp(mllib_image *srcdst, mllib_s32 cmask);
```

Description The `mllib_ImageDivAlpha_Inp()` function divides color channels by the alpha channel on a pixel-by-pixel basis, in place.

For the `MLIB_BYTE` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] / (\text{srcdst}[x][y][a] * 2^{**(-8)})$$

For the `MLIB_SHORT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] / (\text{srcdst}[x][y][a] * 2^{**(-15)})$$

For the `MLIB_USHORT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] / (\text{srcdst}[x][y][a] * 2^{**(-16)})$$

For the `MLIB_INT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] / (\text{srcdst}[x][y][a] * 2^{**(-31)})$$

where `c` and `a` are the indices for the color channels and the alpha channel, respectively, so `c != a`.

In the case of `srcdst[x][y][a] = 0`,

```
srcdst[x][y][c] = 0          if srcdst[x][y][c] = 0  
srcdst[x][y][c] = DATA_TYPE_MAX if srcdst[x][y][c] > 0  
srcdst[x][y][c] = DATA_TYPE_MIN if srcdst[x][y][c] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

cmask Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivAlpha\(3MLIB\)](#), [mllib_ImageDivAlpha_Fp\(3MLIB\)](#),
[mllib_ImageDivAlpha_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivConstShift – division by a constant, with shifting

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageDivConstShift(mllib_image *dst, const mllib_image *src,  
                                     const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageDivConstShift()` function divides each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src[x][y][i] / c[i] * 2^{shift}$$

In the case of $c[i] = 0$,

```
dst[x][y][i] = 0           if src[x][y][i] = 0  
dst[x][y][i] = DATA_TYPE_MAX if src[x][y][i] > 0  
dst[x][y][i] = DATA_TYPE_MIN if src[x][y][i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- c* Constant by which each pixel is divided. `c[i]` contains the constant for channel *i*.
- shift* Left shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivConstShift_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivConstShift_Inp – division by a constant, with shifting

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageDivConstShift_Inp(mllib_image *srcdst,
      const mllib_s32 *c, mllib_s32 shift);
```

Description The `mllib_ImageDivConstShift_Inp()` function divides each pixel in an image by a constant value on a pixel-by-pixel basis, in place. It scales the result by a left shift and writes the result to the image on a pixel-by-pixel basis.

It uses the following equation:

$$srcdst[x][y][i] = srcdst[x][y][i] / c[i] * 2^{shift}$$

In the case of $c[i] = 0$,

$$\begin{aligned} srcdst[x][y][i] &= 0 && \text{if } srcdst[x][y][i] = 0 \\ srcdst[x][y][i] &= DATA_TYPE_MAX && \text{if } srcdst[x][y][i] > 0 \\ srcdst[x][y][i] &= DATA_TYPE_MIN && \text{if } srcdst[x][y][i] < 0 \end{aligned}$$

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

- srcdst* Pointer to source and destination image.
- c* Constant by which each pixel is divided. `c[i]` contains the constant for channel `i`.
- shift* Left shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivConstShift\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDiv_Fp – division

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageDiv_Fp(mllib_image *dst, const mllib_image *src1,
    const mllib_image *src2);
```

Description The `mllib_ImageDiv_Fp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] / src2[x][y][i]$$

where the operation follows the IEEE-754 standard.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src1 Pointer to first source image.

src2 Pointer to second source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDiv1_Fp_Inp\(3MLIB\)](#), [mllib_ImageDiv2_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivShift1_Inp – division with shifting, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDivShift1_Inp(mllib_image *src1dst,
    const mllib_image *src2, mllib_s32 shift);
```

Description The `mllib_ImageDivShift1_Inp()` function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] / \text{src2}[x][y][i] * 2^{\text{shift}}$$

In the case of $\text{src2}[x][y][i] = 0$,

```
src1dst[x][y][i] = 0          if src1dst[x][y][i] = 0
src1dst[x][y][i] = DATA_TYPE_MAX if src1dst[x][y][i] > 0
src1dst[x][y][i] = DATA_TYPE_MIN if src1dst[x][y][i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

src1dst Pointer to first source and destination image.

src2 Pointer to second source image.

shift Left shifting factor. $0 \leq \text{shift} \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivShift\(3MLIB\)](#), [mllib_ImageDivShift2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

REFERENCE

Multimedia Library Functions - Part 3

Name mllib_ImageDivShift2_Inp – division with shifting, in place

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageDivShift2_Inp(mllib_image *src2dst,  
    const mllib_image *src1, mllib_s32 shift);
```

Description The `mllib_ImageDivShift2_Inp()` function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{src2dst}[x][y][i] = \text{src1}[x][y][i] / \text{src2dst}[x][y][i] * 2^{**\text{shift}}$$

In the case of $\text{src2dst}[x][y][i] = 0$,

```
src2dst[x][y][i] = 0           if src1[x][y][i] = 0  
src2dst[x][y][i] = DATA_TYPE_MAX if src1[x][y][i] > 0  
src2dst[x][y][i] = DATA_TYPE_MIN if src1[x][y][i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

src2dst Pointer to second source and destination image.
src1 Pointer to first source image.
shift Left shifting factor. $0 \leq \text{shift} \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivShift\(3MLIB\)](#), [mllib_ImageDivShift1_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageDivShift – division with shifting

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageDivShift(mllib_image *dst, const mllib_image *src1,
                                const mllib_image *src2, mllib_s32 shift);
```

Description The `mllib_ImageDivShift()` function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] / src2[x][y][i] * 2^{**shift}$$

In the case of $src2[x][y][i] = 0$,

```
dst[x][y][i] = 0          if src1[x][y][i] = 0
dst[x][y][i] = DATA_TYPE_MAX if src1[x][y][i] > 0
dst[x][y][i] = DATA_TYPE_MIN if src1[x][y][i] < 0
```

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src1 Pointer to first source image.

src2 Pointer to second source image.

shift Left shifting factor. $0 \leq shift \leq 31$.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageDivShift1_Inp\(3MLIB\)](#), [mllib_ImageDivShift2_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageErode4 – four neighbor erode

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageErode4(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageErode4()` function performs an erode operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 4-neighbor binary images, it uses the following equation:

`dst[x][y][0] = AND{ src[x][y][0],
 src[x-1][y][0], src[x+1][y][0],
 src[x][y-1][0], src[x][y+1][0] }`

For 4-neighbor grayscale images, it uses the following equation:

`dst[x][y][0] = MIN{ src[x][y][0],
 src[x-1][y][0], src[x+1][y][0],
 src[x][y-1][0], src[x][y+1][0] }`

where `x = 1, ..., w-2`; `y = 1, ..., h-2`.

Parameters The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageErode4_Fp\(3MLIB\)](#), [mllib_ImageErode8\(3MLIB\)](#),
[mllib_ImageErode8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageErode4_Fp – four neighbor erode

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageErode4_Fp(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageErode4_Fp()` function performs an erode operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 4-neighbor grayscale images, it uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[x][y][0], \\ \text{src}[x-1][y][0], \text{src}[x+1][y][0], \\ \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageErode4\(3MLIB\)](#), [mllib_ImageErode8\(3MLIB\)](#), [mllib_ImageErode8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageErode8 – eight neighbor erode

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageErode8(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageErode8()` function performs an erode operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 8-neighbor binary images, it uses the following equation:

$$dst[x][y][0] = AND\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

For 8-neighbor grayscale images, it uses the following equation:

$$dst[x][y][0] = MIN\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageErode4\(3MLIB\)](#), [mllib_ImageErode4_Fp\(3MLIB\)](#),
[mllib_ImageErode8_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageErode8_Fp – eight neighbor erode

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageErode8_Fp(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageErode8_Fp()` function performs an erode operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 8-neighbor grayscale images, it uses the following equation:

$$dst[x][y][0] = \min\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where $x = 1, \dots, w-2$; $y = 1, \dots, h-2$.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageErode4\(3MLIB\)](#), [mllib_ImageErode4_Fp\(3MLIB\)](#), [mllib_ImageErode8\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageExp – computes the exponent of the image pixels

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageExp(mllib_image *dst, const mllib_image *src);
```

Description The mllib_ImageExp() function computes the exponent of the image pixels.

It uses the following equation:

$$dst[x][y][i] = e^{**}src[x][y][i]$$

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageExp_Fp\(3MLIB\)](#), [mllib_ImageExp_Fp_Inp\(3MLIB\)](#), [mllib_ImageExp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageExp_Fp – computes the exponent of the image pixels

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageExp_Fp(mllib_image *dst, const mllib_image *src);
```

Description The `mllib_ImageExp_Fp()` function computes the exponent of the floating-point image pixels.

It uses the following equation:

$$\text{dst}[x][y][i] = e^{**}\text{src}[x][y][i]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageExp\(3MLIB\)](#), [mllib_ImageExp_Fp_Inp\(3MLIB\)](#), [mllib_ImageExp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageExp_Fp_Inp – computes the exponent of the image pixels

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageExp_Fp_Inp(mllib_image *srcdst);`

Description The `mllib_ImageExp_Fp_Inp()` function computes the exponent of the floating-point image pixels.

It uses the following equation:

$$srcdst[x][y][i] = e^{**srcdst[x][y][i]}$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageExp\(3MLIB\)](#), [mllib_ImageExp_Fp\(3MLIB\)](#), [mllib_ImageExp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

- Name** mllib_ImageExp_Inp – computes the exponent of the image pixels
- Synopsis**

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageExp_Inp(mllib_image *srcdst);
```
- Description** The mllib_ImageExp_Inp() function computes the exponent of the image pixels, in place. It uses the following equation:
$$srcdst[x][y][i] = e^{**srcdst[x][y][i]}$$
- Parameters** The function takes the following arguments:
srcdst Pointer to source and destination image.
- Return Values** The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
- Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

- See Also** [mllib_ImageExp\(3MLIB\)](#), [mllib_ImageExp_Fp\(3MLIB\)](#), [mllib_ImageExp_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageExtrema2, mllib_ImageExtrema2_Fp – image extrema

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageExtrema2(mllib_s32 *min, mllib_s32 *max,  
    const mllib_image *img, mllib_s32 xStart, mllib_s32 yStart,  
    mllib_s32 xPeriod, mllib_s32 yPeriod);  
  
mllib_status mllib_ImageExtrema2_Fp(mllib_d64 *min, mllib_d64 *max,  
    const mllib_image *img, mllib_s32 xStart, mllib_s32 yStart,  
    mllib_s32 xPeriod, mllib_s32 yPeriod);
```

Description Each of the functions determines the extrema values for each channel in an image, possibly with subsampling.

It uses the following equation:

```
min[i] = MIN{ img[x][y][i] }  
max[i] = MAX{ img[x][y][i] }
```

where

```
x = xStart + p*xPeriod; 0 ≤ p < (w - xStart)/xPeriod  
y = yStart + q*yPeriod; 0 ≤ q < (h - yStart)/yPeriod
```

Parameters Each of the functions takes the following arguments:

- min* Pointer to minimum vector, where length is the number of channels in the image. min[i] contains the minimum of channel i.
- max* Pointer to maximum vector, where length is the number of channels in the image. max[i] contains the maximum of channel i.
- img* Pointer to a source image.
- xStart* Initial X sample coordinate.
- yStart* Initial Y sample coordinate.
- xPeriod* X sample rate. xPeriod ≥ 1.
- yPeriod* Y sample rate. yPeriod ≥ 1.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `mllib_ImageExtremaLocations(3MLIB), mllib_ImageMaximum(3MLIB),
mllib_ImageMaximum_Fp(3MLIB), mllib_ImageMinimum(3MLIB),
mllib_ImageMinimum_Fp(3MLIB), attributes(5)`

Name mllib_ImageExtremaLocations, mllib_ImageExtremaLocations_Fp – image extrema and their locations

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageExtremaLocations( mllib_s32 *min, mllib_s32 *max,  
    const mllib_image *img, mllib_s32 xStart, mllib_s32 yStart,  
    mllib_s32 xPeriod, mllib_s32 yPeriod, mllib_s32 saveLocations,  
    mllib_s32 maxRuns, mllib_s32 *minCounts, mllib_s32 *maxCounts,  
    mllib_s32 **minLocations, mllib_s32 **maxLocations, mllib_s32 len);  
  
mllib_status mllib_ImageExtremaLocations_Fp( mllib_d64 *min, mllib_d64 *max,  
    const mllib_image *img, mllib_s32 xStart, mllib_s32 yStart,  
    mllib_s32 xPeriod, mllib_s32 yPeriod, mllib_s32 saveLocations,  
    mllib_s32 maxRuns, mllib_s32 *minCounts, mllib_s32 *maxCounts,  
    mllib_s32 **minLocations, mllib_s32 **maxLocations, mllib_s32 len);
```

Description Each of the functions finds the image-wise minimum and maximum pixel values for each channel, and optionally, their locations.

Each of the functions scans an image, finds the minimum and maximum pixel values for each channel, and finds the locations of those pixels with the minimum or maximum values.

The user provides initial minimum/maximum values through the arguments `min` and `max`. This function will update them based on findings.

The set of pixels scanned may furthermore be reduced by specifying `xPeriod` and `yPeriod` parameters that specify the sampling rate along each axis.

The set of pixels to be scanned may be obtained from the following equation:

$$\begin{aligned}x &= xStart + p \cdot xPeriod; & 0 \leq p < (w - xStart)/xPeriod \\y &= yStart + q \cdot yPeriod; & 0 \leq q < (h - yStart)/yPeriod\end{aligned}$$

The locations of the minimum/maximum, if asked, are recorded in a format of run-length coding. Each run-length code, or simply called a run, has a format of (`xStart`, `yStart`, `length`). Here `length` is defined on the low-resolution image (with downsampling factors of $1/xPeriod$, $1/yPeriod$) and does not cross rows. So the run-length code (`xStart`, `yStart`, `length`) means that the pixels at (`xStart`, `yStart`), (`xStart` + `xPeriod`, `yStart`), ..., (`xStart` + (`length` - 1) * `xPeriod`, `yStart`) of the original image have a value of the minimum/maximum.

The buffers for `minLocations` and `maxLocations` are organized in the following format for each channel `i`:

```
minLocations[i][0] = xStart0; // the 1st run  
minLocations[i][1] = yStart0;  
minLocations[i][2] = length0;  
minLocations[i][3] = xStart1; // the 2nd run
```

```
minLocations[i][4] = yStart1;
minLocations[i][5] = length1;
        ..... // more runs
minLocations[i][len-1] = ...;
```

It is the user's responsibility to allocate enough memory for the buffers for `minLocations` and `maxLocations`. This function may return `MLIB_OUTOFRANGE`, if any of the buffers is not big enough.

Parameters The function takes the following arguments:

- min* Pointer to the minimum values.
- max* Pointer to the maximum values.
- img* Pointer to the input image.
- xStart* Initial X sample coordinate.
- yStart* Initial Y sample coordinate.
- xPeriod* X sampling rate. $xPeriod \geq 1$.
- yPeriod* Y sampling rate. $yPeriod \geq 1$.
- saveLocations* If true (i.e., `saveLocations != 0`), find the extrema locations; otherwise only find the extrema.
- maxRuns* Number of runs of the minimum/maximum the caller expects for each channel. $maxRuns \geq 1$. If it is `MLIB_S32_MAX`, all the minimum/maximum locations should be recorded.
- minCounts* Pointer to the numbers of runs of the minimum recorded in `minLocations`.
- maxCounts* Pointer to the numbers of runs of the maximum recorded in `maxLocations`.
- minLocations* Pointer to the minimum locations in a format of run-length coding.
- maxLocations* Pointer to the maximum locations in a format of run-length coding.
- len* Length of the buffers for the minimum/maximum locations in each channel.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageExtrema2(3MLIB)`, `mllib_ImageMaximum(3MLIB)`,
`mllib_ImageMaximum_Fp(3MLIB)`, `mllib_ImageMinimum(3MLIB)`,
`mllib_ImageMinimum_Fp(3MLIB)`, `attributes(5)`

Name mllib_ImageFilteredSubsample, mllib_ImageFilteredSubsample_Fp – antialias filters and subsamples an image

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageFilteredSubsample(mllib_image *dst,
    const mllib_image *src, mllib_s32 scaleX, mllib_s32 scaleY,
    mllib_s32 transX, mllib_s32 transY,
    const mllib_d64 *hKernel, const mllib_d64 *vKernel, mllib_s32 hSize,
    mllib_s32 vSize, mllib_s32 hParity, mllib_s32 vParity, mllib_edge edge);

mllib_status mllib_ImageFilteredSubsample_Fp(mllib_image *dst,
    const mllib_image *src, mllib_s32 scaleX, mllib_s32 scaleY,
    mllib_s32 transX, mllib_s32 transY,
    const mllib_d64 *hKernel, const mllib_d64 *vKernel, mllib_s32 hSize,
    mllib_s32 vSize, mllib_s32 hParity, mllib_s32 vParity, mllib_edge edge);
```

Description Each of the functions antialias filters and subsamples an image.

The effect of one of the functions on an image is equivalent to performing convolution (filter) followed by subsampling (zoom out).

The functions are similar to the `mllib_ImageZoomTranslate()` and `mllib_ImageZoomTranslate_Fp()` functions. But they have different definitions on scale factors and translations, hence use different coordinate mapping equations. The `scaleX` and `scaleY` used by `mllib_ImageFilteredSubsample()` and `mllib_ImageFilteredSubsample_Fp()` are the reciprocals of the `zoomx` and `zoomy`, respectively, used by `mllib_ImageZoomTranslate()` and `mllib_ImageZoomTranslate_Fp()`.

The functions use the following equations for coordinate mapping:

$$xS = xD * scaleX + transX$$

$$yS = yD * scaleY + transY$$

where, a point (xD, yD) in the destination image is backward mapped to a point (xS, yS) in the source image. The arguments `transX` and `transY` are provided to support tiling.

The subsample terms, i.e., the scale factors `scaleX` and `scaleY`, are restricted to positive integral values. Geometrically, one destination pixel maps to `scaleX` by `scaleY` source pixels. With odd scale factors, destination pixel centers map directly onto source pixel centers. With even scale factors, destination pixel centers map squarely between source pixel centers. Below are examples of even, odd, and combination cases.

s	s	s	s	s	s	s	s	s	s	s	s
	d		d		d						
s	s	s	s	s	s	s	d	s	s	d	s
s	s	s	s	s	s	s	s	s	s	s	s
	d		d		d						

s s s s s s	s s s s s s
s s s s s s	s d s s d s
d d d	
s s s s s s	s s s s s s
Even scaleX/Y factors	Odd scaleX/Y factors
s s s s s s	s s s s s s
d d	
s s s s s s	s d s s d s s d s
s s s s s s	s s s s s s
d d	
s s s s s s	s s s s s s
s s s s s s	s d s s d s s d s
d d	
s s s s s s	s s s s s s
Odd/even scaleX/Y factors	Even/odd scaleX/Y factors

where

s = source pixel centers
d = destination pixel centers mapped to source

The applied filter is quadrant symmetric (typically antialias + resample). The filter is product-separable, quadrant symmetric, and is defined by half of its span. Parity is used to signify whether the symmetric kernel has a double center (even parity) or a single center value (odd parity). For example, if `hParity == 0` (even), the horizontal kernel is defined as:

```
hKernel[hSize-1], ..., hKernel[0], hKernel[0], ...,
hKernel[hSize-1]
```

Otherwise, if `hParity == 1` (odd), the horizontal kernel is defined as:

```
hKernel[hSize-1], ..., hKernel[0], ...,
hKernel[hSize-1]
```

Horizontal and vertical kernels representing convolved resample (i.e., the combined separable kernels) can be computed from a convolution filter (with odd parity), a resample filter, and because the subsample factors affect resample weights, the subsample scale factors. It is the user's responsibility to provide meaningful combined kernels.

To compute the value of a pixel centered at point (xD, yD) in the destination image, apply the combined kernel to the source image by aligning the kernel's geometric center to the backward mapped point (xS, yS) in the source image. In the cases that it can not be exactly

on top of point (xS, yS), the kernel's center should be half-pixel right and/or below that point. When this is done in a separable manner, the centers of horizontal and vertical kernels should align with xS and yS, respectively.

The combination of subsampling and filtering has performance benefits over sequential fucntion usage in part due to the symmetry constraints imposed by only allowing integer parameters for scaling and only allowing separable symmetric filters.

- Parameters
- The function takes the following arguments:
- dst*

Pointer to destination image.
- src*

Pointer to source image.
- scaleX*

The x scale factor of subsampling.
- scaleY*

The y scale factor of subsampling.
- transX*

The x translation.
- transY*

The y translation.
- hKernel*

Pointer to the compact form of horizontal kernel.
- vKernel*

Pointer to the compact form of vertical kernel.
- hSize*

Size of array *hKernel*.
- vSize*

Size of array *vKernel*.
- hParity*

Parity of horizontal kernel (0: even, 1: odd).
- vParity*

Parity of vertical kernel (0: even, 1: odd).
- edge*

Type of edge condition. It can be one of the following:
MLIB_EDGE_DST_NO_WRITE

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also

[mllib_ImageSubsampleAverage\(3MLIB\)](#), [mllib_ImageZoomTranslate\(3MLIB\)](#), [mllib_ImageZoomTranslate_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageFlipAntiDiag – anti-diagonal flip

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageFlipAntiDiag(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageFlipAntiDiag()` function flips an image on the anti-diagonal axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTEVALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageFlipAntiDiag_Fp(3MLIB)`, `mllib_ImageFlipMainDiag(3MLIB)`,
`mllib_ImageFlipMainDiag_Fp(3MLIB)`, `mllib_ImageFlipX(3MLIB)`,
`mllib_ImageFlipX_Fp(3MLIB)`, `mllib_ImageFlipY(3MLIB)`, `mllib_ImageFlipY_Fp(3MLIB)`,
`mllib_ImageRotate90(3MLIB)`, `mllib_ImageRotate90_Fp(3MLIB)`,
`mllib_ImageRotate180(3MLIB)`, `mllib_ImageRotate180_Fp(3MLIB)`,
`mllib_ImageRotate270(3MLIB)`, `mllib_ImageRotate270_Fp(3MLIB)`, [attributes\(5\)](#)

Name mlib_ImageFlipAntiDiag_Fp – anti-diagonal flip

Synopsis

```
cc [ flag... ] file... -llib [ library... ]
#include <mlib.h>
```

```
mlib_status mlib_ImageFlipAntiDiag_Fp(mlib_image *dst,
    const mlib_image *src);
```

Description The `mlib_ImageFlipAntiDiag_Fp()` function flips a floating-point image on the anti-diagonal axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageFlipAntiDiag\(3MLIB\)](#), [mlib_ImageFlipMainDiag\(3MLIB\)](#), [mlib_ImageFlipMainDiag_Fp\(3MLIB\)](#), [mlib_ImageFlipX\(3MLIB\)](#), [mlib_ImageFlipX_Fp\(3MLIB\)](#), [mlib_ImageFlipY\(3MLIB\)](#), [mlib_ImageFlipY_Fp\(3MLIB\)](#), [mlib_ImageRotate90\(3MLIB\)](#), [mlib_ImageRotate90_Fp\(3MLIB\)](#), [mlib_ImageRotate180\(3MLIB\)](#), [mlib_ImageRotate180_Fp\(3MLIB\)](#), [mlib_ImageRotate270\(3MLIB\)](#), [mlib_ImageRotate270_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageFlipMainDiag – main diagonal flip

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageFlipMainDiag(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageFlipMainDiag()` function flips an image on the main diagonal.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTEVALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageFlipAntiDiag(3MLIB)`, `mllib_ImageFlipAntiDiag_Fp(3MLIB)`,
`mllib_ImageFlipMainDiag_Fp(3MLIB)`, `mllib_ImageFlipX(3MLIB)`,
`mllib_ImageFlipX_Fp(3MLIB)`, `mllib_ImageFlipY(3MLIB)`, `mllib_ImageFlipY_Fp(3MLIB)`,
`mllib_ImageRotate90(3MLIB)`, `mllib_ImageRotate90_Fp(3MLIB)`,
`mllib_ImageRotate180(3MLIB)`, `mllib_ImageRotate180_Fp(3MLIB)`,
`mllib_ImageRotate270(3MLIB)`, `mllib_ImageRotate270_Fp(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageFlipMainDiag_Fp – main diagonal flip

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageFlipMainDiag_Fp(mllib_image *dst,
    const mllib_image *src);
```

Description The `mllib_ImageFlipMainDiag_Fp()` function flips a floating-point image on the main diagonal.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageFlipAntiDiag\(3MLIB\)](#), [mllib_ImageFlipAntiDiag_Fp\(3MLIB\)](#), [mllib_ImageFlipMainDiag\(3MLIB\)](#), [mllib_ImageFlipX\(3MLIB\)](#), [mllib_ImageFlipX_Fp\(3MLIB\)](#), [mllib_ImageFlipY\(3MLIB\)](#), [mllib_ImageFlipY_Fp\(3MLIB\)](#), [mllib_ImageRotate90\(3MLIB\)](#), [mllib_ImageRotate90_Fp\(3MLIB\)](#), [mllib_ImageRotate180\(3MLIB\)](#), [mllib_ImageRotate180_Fp\(3MLIB\)](#), [mllib_ImageRotate270\(3MLIB\)](#), [mllib_ImageRotate270_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageFlipX – X-axis flip

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageFlipX(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageFlipX()` function flips an image on its X axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTEVALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageFlipAntiDiag(3MLIB)`, `mllib_ImageFlipAntiDiag_Fp(3MLIB)`,
`mllib_ImageFlipMainDiag(3MLIB)`, `mllib_ImageFlipMainDiag_Fp(3MLIB)`,
`mllib_ImageFlipX_Fp(3MLIB)`, `mllib_ImageFlipY(3MLIB)`, `mllib_ImageFlipY_Fp(3MLIB)`,
`mllib_ImageRotate90(3MLIB)`, `mllib_ImageRotate90_Fp(3MLIB)`,
`mllib_ImageRotate180(3MLIB)`, `mllib_ImageRotate180_Fp(3MLIB)`,
`mllib_ImageRotate270(3MLIB)`, `mllib_ImageRotate270_Fp(3MLIB)`, [attributes\(5\)](#)

Name mlib_ImageFlipX_Fp – X-axis flip

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageFlipX_Fp(mlib_image *dst, const mlib_image *src);
```

Description The `mlib_ImageFlipX_Fp()` function flips a floating-point image on its X axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageFlipAntiDiag\(3MLIB\)](#), [mlib_ImageFlipAntiDiag_Fp\(3MLIB\)](#),
[mlib_ImageFlipMainDiag\(3MLIB\)](#), [mlib_ImageFlipMainDiag_Fp\(3MLIB\)](#),
[mlib_ImageFlipX\(3MLIB\)](#), [mlib_ImageFlipY\(3MLIB\)](#), [mlib_ImageFlipY_Fp\(3MLIB\)](#),
[mlib_ImageRotate90\(3MLIB\)](#), [mlib_ImageRotate90_Fp\(3MLIB\)](#),
[mlib_ImageRotate180\(3MLIB\)](#), [mlib_ImageRotate180_Fp\(3MLIB\)](#),
[mlib_ImageRotate270\(3MLIB\)](#), [mlib_ImageRotate270_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageFlipY – Y-axis flip

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageFlipY(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageFlipY()` function flips an image on its Y axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageFlipAntiDiag(3MLIB)`, `mllib_ImageFlipAntiDiag_Fp(3MLIB)`,
`mllib_ImageFlipMainDiag(3MLIB)`, `mllib_ImageFlipMainDiag_Fp(3MLIB)`,
`mllib_ImageFlipX(3MLIB)`, `mllib_ImageFlipX_Fp(3MLIB)`, `mllib_ImageFlipY_Fp(3MLIB)`,
`mllib_ImageRotate90(3MLIB)`, `mllib_ImageRotate90_Fp(3MLIB)`,
`mllib_ImageRotate180(3MLIB)`, `mllib_ImageRotate180_Fp(3MLIB)`,
`mllib_ImageRotate270(3MLIB)`, `mllib_ImageRotate270_Fp(3MLIB)`, [attributes\(5\)](#)

Name mlib_ImageFlipY_Fp – Y-axis flip

Synopsis `cc [flag...] file... -lmlib [library...]`
`#include <mlib.h>`

```
mlib_status mlib_ImageFlipY_Fp(mlib_image *dst, const mlib_image *src);
```

Description The `mlib_ImageFlipY_Fp()` function flips a floating-point image on its Y axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageFlipAntiDiag\(3MLIB\)](#), [mlib_ImageFlipAntiDiag_Fp\(3MLIB\)](#),
[mlib_ImageFlipMainDiag\(3MLIB\)](#), [mlib_ImageFlipMainDiag_Fp\(3MLIB\)](#),
[mlib_ImageFlipX\(3MLIB\)](#), [mlib_ImageFlipX_Fp\(3MLIB\)](#), [mlib_ImageFlipY\(3MLIB\)](#),
[mlib_ImageRotate90\(3MLIB\)](#), [mlib_ImageRotate90_Fp\(3MLIB\)](#),
[mlib_ImageRotate180\(3MLIB\)](#), [mlib_ImageRotate180_Fp\(3MLIB\)](#),
[mlib_ImageRotate270\(3MLIB\)](#), [mlib_ImageRotate270_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageFourierTransform – Fourier transform

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageFourierTransform(mllib_image *dst,
    const mllib_image *src, mllib_fourier_mode mode);
```

Description The `mllib_ImageFourierTransform()` function performs a two-dimensional Fourier transformation. The source and destination images must be the same type and the same size. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`. The height and width of the images must be some positive power of 2 (but they do not have to be equal).

They can have 1 or 2 channels. If the source image has just one channel the imaginary parts are assumed to be zero. If the destination image has just one channel, then it is assumed that the imaginary parts of the output can be discarded. But in case both source and destination images are one-channel images, then `MLIB_FAILURE` is returned.

The predefined modes used in the image Fourier transform function are as follows:

Mode	Description
<code>MLIB_DFT_SCALE_NONE</code>	Forward DFT without scaling
<code>MLIB_DFT_SCALE_MXN</code>	Forward DFT with scaling of $1/(M*N)$
<code>MLIB_DFT_SCALE_SQRT</code>	Forward DFT with scaling of $1/\sqrt{M*N}$
<code>MLIB_IDFT_SCALE_NONE</code>	Inverse DFT without scaling
<code>MLIB_IDFT_SCALE_MXN</code>	Inverse DFT with scaling of $1/(M*N)$
<code>MLIB_IDFT_SCALE_SQRT</code>	Inverse DFT with scaling of $1/\sqrt{M*N}$

Parameters The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
mode Mode of the transform.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also [attributes\(5\)](#)

Name mllib_ImageGetBitOffset – get bitoffset

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_s32 mllib_ImageGetBitOffset(const mllib_image *img);`

Description A query function that returns the bitoffset public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to a mediaLib image structure.

Return Values The function returns the offset, in terms of bits, of an image from the beginning of the data buffer to the first pixel.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetChannels\(3MLIB\)](#), [mllib_ImageGetData\(3MLIB\)](#),
[mllib_ImageGetFlags\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetChannels – get channels

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_s32 mllib_ImageGetChannels(const mllib_image *img);`

Description A query function that returns the channels public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to a mediaLib image structure.

Return Values The function returns the number of channels in an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetData\(3MLIB\)](#),
[mllib_ImageGetFlags\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetData – get data

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
void *mllib_ImageGetData(const mllib_image *img);
```

Description The `mllib_ImageGetData()` function returns the data public field of a mediaLib image structure. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values The function returns a pointer to the image data.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetFlags\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetFlags – get flags

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_s32 mllib_ImageGetFlags(const mllib_image *img);
```

Description The `mllib_ImageGetFlags()` function returns the attribute flags of an image. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values The function returns the value of the attribute flags.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetFormat – get format

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_format mllib_ImageGetFormat(const mllib_image *img);`

Description A query function that returns the format public field of a mllib_image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to a mediaLib image structure.

Return Values The function returns the value of the format of an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageSetFormat\(3MLIB\)](#), [mllib_ImageGetBitOffset\(3MLIB\)](#),
[mllib_ImageGetChannels\(3MLIB\)](#), [mllib_ImageGetData\(3MLIB\)](#),
[mllib_ImageGetFlags\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetHeight – get height

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_s32 mllib_ImageGetHeight(const mllib_image *img);
```

Description A query function that returns the height public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values The function returns the value of the height (in pixels) of an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetFlags\(3MLIB\)](#),
[mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetPaddings – get paddings

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]  
#include <mllib.h>
```

```
mllib_u8 *mllib_ImageGetPaddings(const mllib_image *img);
```

Description A query function that returns the borders public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to a mediaLib image structure.

Return Values The function returns a pointer to the image paddings. paddings[0] holds leftPadding; paddings[1] holds topPadding; paddings[2] holds rightPadding; paddings[3] holds bottomPadding.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetFlags\(3MLIB\)](#),
[mllib_ImageGetHeight\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#),
[mllib_ImageSetPaddings\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetStride – get stride

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_s32 mllib_ImageGetStride(const mllib_image *img);`

Description A query function that returns the stride public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:
img Pointer to source image.

Return Values The function returns the value of the stride (in bytes) of an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetFlags\(3MLIB\)](#),
[mllib_ImageGetHeight\(3MLIB\)](#), [mllib_ImageGetPaddings\(3MLIB\)](#),
[mllib_ImageGetType\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetType – get type

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_type mllib_ImageGetType(const mllib_image *img);`

Description A query function that returns the type public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values The function returns the value of the type of an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#),
[mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetFlags\(3MLIB\)](#),
[mllib_ImageGetHeight\(3MLIB\)](#), [mllib_ImageGetPaddings\(3MLIB\)](#),
[mllib_ImageGetStride\(3MLIB\)](#), [mllib_ImageGetWidth\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGetWidth – get width

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_s32 mllib_ImageGetWidth(const mllib_image *img);`

Description A query function that returns the width public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values The function returns the value of the width of an image.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGetBitOffset\(3MLIB\)](#), [mllib_ImageGetChannels\(3MLIB\)](#), [mllib_ImageGetData\(3MLIB\)](#), [mllib_ImageGetFlags\(3MLIB\)](#), [mllib_ImageGetHeight\(3MLIB\)](#), [mllib_ImageGetPaddings\(3MLIB\)](#), [mllib_ImageGetStride\(3MLIB\)](#), [mllib_ImageGetType\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGradient3x3 – 3x3 gradient filter

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageGradient3x3(mllib_image *dst, const mllib_image *src,  
    const mlib_d64 *hmask, const mlib_d64 *vmask, mlib_s32 cmask, mlib_edge edge);
```

Description The `mllib_ImageGradient3x3()` function performs edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using 3x3 gradient filtering.

It uses the following equation:

$$dst[x][y][i] = (SH(x,y,i)**2 + SV(x,y,i)**2)**0.5$$

where `SH()` and `SV()` are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hmask* Pointer to horizontal mask in row-major order.
- vmask* Pointer to vertical mask in row-major order.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
 - MLIB_EDGE_DST_NO_WRITE
 - MLIB_EDGE_DS_FILL_ZERO
 - MLIB_EDGE_DST_COPY_SRC
 - MLIB_EDGE_SR_EXTEND

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageGradient3x3_Fp(3MLIB)`, `mlib_ImageGradientMxN(3MLIB)`,
`mlib_ImageGradientMxN_Fp(3MLIB)`, `attributes(5)`

Name mllib_ImageGradient3x3_Fp – 3x3 gradient filter

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageGradient3x3_Fp(mllib_image *dst,  
    const mllib_image *src, const mllib_d64 *hmask,  
    const mllib_d64 *vmask, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageGradient3x3_Fp()` function performs floating-point edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using 3x3 gradient filtering.

It uses the following equation:

$$dst[x][y][i] = (SH(x,y,i)**2 + SV(x,y,i)**2)**0.5$$

where `SH()` and `SV()` are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hmask* Pointer to horizontal mask in row-major order.
- vmask* Pointer to vertical mask in row-major order.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:

 MLIB_EDGE_DST_NO_WRITE
 MLIB_EDGE_DS_FILL_ZERO
 MLIB_EDGE_DST_COPY_SRC
 MLIB_EDGE_SR_EXTEND

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageGradient3x3(3MLIB)`, `mllib_ImageGradientMxN(3MLIB)`,
`mllib_ImageGradientMxN_Fp(3MLIB)`, `attributes(5)`

Name mllib_ImageGradientMxN – MxN gradient filter

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageGradientMxN(mllib_image *dst, const mllib_image *src,  
    const mllib_d64 *hmask, const mllib_d64 *vmask, mllib_s32 m, mllib_s32 n,  
    mllib_s32 dm, mllib_s32 dn, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageGradientMxN()` function performs edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using MxN gradient filtering.

It uses the following equation:

$$dst[x][y][i] = (SH(x,y,i)**2 + SV(x,y,i)**2)**0.5$$

where `SH()` and `SV()` are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>hmask</i>	Pointer to horizontal mask in row-major order.
<i>vmask</i>	Pointer to vertical mask in row-major order.
<i>m</i>	Width of the convolution kernel. $m > 1$.
<i>n</i>	Height of the convolution kernel. $n > 1$.
<i>dm</i>	X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
<i>cmask</i>	Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_DS_FILL_ZERO MLIB_EDGE_DST_COPY_SRC MLIB_EDGE_SR_EXTEND

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGradientMxN_Fp\(3MLIB\)](#), [mllib_ImageGradient3x3\(3MLIB\)](#),
[mllib_ImageGradient3x3_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGradientMxN_Fp – MxN gradient filter

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageGradientMxN_Fp(mllib_image *dst,  
    const mllib_image *src, const mllib_d64 *hmask,  
    const mllib_d64 *vmask, mllib_s32 m, mllib_s32 n, mllib_s32 dm,  
    mllib_s32 dn, mllib_s32 cmask, mllib_edge edge);
```

Description The `mllib_ImageGradientMxN_Fp()` function performs floating-point edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using MxN gradient filtering.

It uses the following equation:

$$\text{dst}[x][y][i] = (\text{SH}(x,y,i)**2 + \text{SV}(x,y,i)**2)**0.5$$

where `SH()` and `SV()` are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

Parameters The function takes the following arguments:

<i>dst</i>	Pointer to destination image.
<i>src</i>	Pointer to source image.
<i>hmask</i>	Pointer to horizontal mask in row-major order.
<i>vmask</i>	Pointer to vertical mask in row-major order.
<i>m</i>	Width of the convolution kernel. $m > 1$.
<i>n</i>	Height of the convolution kernel. $n > 1$.
<i>dm</i>	X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
<i>dn</i>	Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
<i>cmask</i>	Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_DS_FILL_ZERO MLIB_EDGE_DST_COPY_SRC MLIB_EDGE_SR_EXTEND

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGradientMxN\(3MLIB\)](#), [mllib_ImageGradient3x3\(3MLIB\)](#),
[mllib_ImageGradient3x3_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGridWarp – grid-based image warp

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageGridWarp(mllib_image *dst, const mllib_image *src,  
    const mllib_f32 *xWarpPos, const mllib_f32 *yWarpPos,  
    mllib_d64 postShiftX, mllib_d64 postShiftY,  
    mllib_s32 xStart, mllib_s32 xStep, mllib_s32 xNumCells,  
    mllib_s32 yStart, mllib_s32 yStep, mllib_s32 yNumCells,  
    mllib_filter filter, mllib_edge edge);
```

Description The `mllib_ImageGridWarp()` function performs a regular grid-based image warp. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

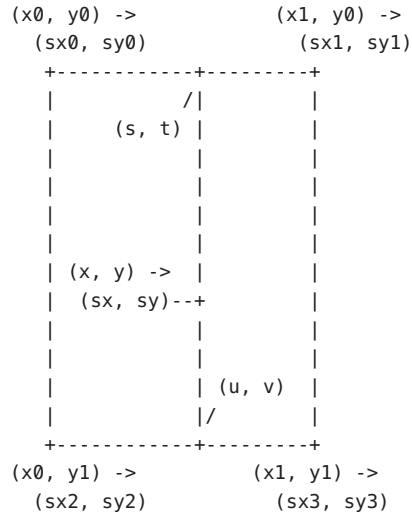
For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

```
xfrac = (x - x0)/(x1 - x0)  
yfrac = (y - y0)/(y1 - y0)  
  
s = sx0 + (sx1 - sx0)*xfrac  
t = sy0 + (sy1 - sy0)*xfrac  
  
u = sx2 + (sx3 - sx2)*xfrac  
v = sy2 + (sy3 - sy2)*xfrac  
  
sx = s + (u - s)*yfrac - postShiftX  
sy = t + (v - t)*yfrac - postShiftY
```

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:



The results of above interpolation are shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at (xStart, yStart). Each cell has width equal to xStep and height equal to yStep, and there are xNumCells cells horizontally and yNumCells cells vertically.

The degree of warping within each cell is defined by the values in xWarpPos and yWarpPos parameters. Each of these parameters must contain (xNumCells + 1)*(yNumCells + 1) values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose xNumCells is equal to 2 and yNumCells is equal to 1. Then the order of the data in the xWarpPos would be:

x00, x10, x20, x01, x11, x21

and in the yWarpPos:

y00, y10, y20, y01, y11, y21

for a total of (2 + 1)*(1 + 1) = 6 elements in each table.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

<i>xWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing horizontal warp positions at the grid points, in row-major order.
<i>yWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing vertical warp positions at the grid points, in row-major order.
<i>postShiftX</i>	The displacement to apply to source X positions.
<i>postShiftY</i>	The displacement to apply to source Y positions.
<i>xStart</i>	The minimum X coordinate of the grid.
<i>xStep</i>	The horizontal spacing between grid cells.
<i>xNumCells</i>	The number of grid cell columns.
<i>yStart</i>	The minimum Y coordinate of the grid.
<i>yStep</i>	The vertical spacing between grid cells.
<i>yNumCells</i>	The number of grid cell rows.
<i>filter</i>	Type of resampling filter. It can be one of the following: MLIB_NEAREST MLIB_BILINEAR MLIB_BICUBIC MLIB_BICUBIC2
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_SRC_PADDED

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGridWarp_Fp\(3MLIB\)](#), [mllib_ImageGridWarpTable\(3MLIB\)](#), [mllib_ImageGridWarpTable_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGridWarp_Fp – grid-based image warp

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
mllib_status mllib_ImageGridWarp_Fp(mllib_image *dst, const mllib_image *src,
    const mllib_f32 *xWarpPos, const mllib_f32 *yWarpPos,
    mllib_d64 postShiftX, mllib_d64 postShiftY,
    mllib_s32 xStart, mllib_s32 xStep, mllib_s32 xNumCells,
    mllib_s32 yStart, mllib_s32 yStep, mllib_s32 yNumCells,
    mllib_filter filter, mllib_edge edge);
```

Description The `mllib_ImageGridWarp_Fp()` function performs a regular grid-based image warp on a floating-point image. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

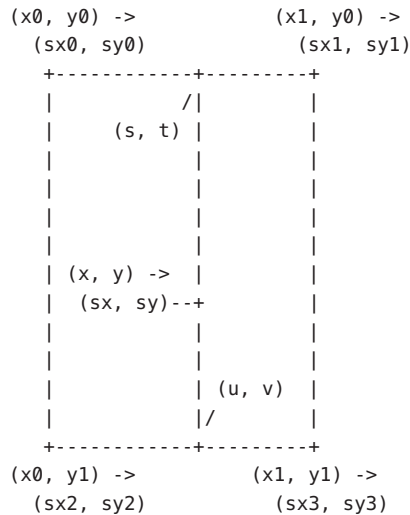
```
xfrac = (x - x0)/(x1 - x0)
yfrac = (y - y0)/(y1 - y0)

s = sx0 + (sx1 - sx0)*xfrac
t = sy0 + (sy1 - sy0)*xfrac

u = sx2 + (sx3 - sx2)*xfrac
v = sy2 + (sy3 - sy2)*xfrac

sx = s + (u - s)*yfrac - postShiftX
sy = t + (v - t)*yfrac - postShiftY
```

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:



The results of above interpolation are shifted by `(-postShiftX, -postShiftY)` to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at `(xStart, yStart)`. Each cell has width equal to `xStep` and height equal to `yStep`, and there are `xNumCells` cells horizontally and `yNumCells` cells vertically.

The degree of warping within each cell is defined by the values in `xWarpPos` and `yWarpPos` parameters. Each of these parameters must contain $(xNumCells + 1) * (yNumCells + 1)$ values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose `xNumCells` is equal to 2 and `yNumCells` is equal to 1. Then the order of the data in the `xWarpPos` would be:

`x00, x10, x20, x01, x11, x21`

and in the `yWarpPos`:

`y00, y10, y20, y01, y11, y21`

for a total of $(2 + 1) * (1 + 1) = 6$ elements in each table.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

<i>xWarpPos</i>	A float array of length (xNumCells + 1)*(yNumCells + 1) containing horizontal warp positions at the grid points, in row-major order.
<i>yWarpPos</i>	A float array of length (xNumCells + 1)*(yNumCells + 1) containing vertical warp positions at the grid points, in row-major order.
<i>postShiftX</i>	The displacement to apply to source X positions.
<i>postShiftY</i>	The displacement to apply to source Y positions.
<i>xStart</i>	The minimum X coordinate of the grid.
<i>xStep</i>	The horizontal spacing between grid cells.
<i>xNumCells</i>	The number of grid cell columns.
<i>yStart</i>	The minimum Y coordinate of the grid.
<i>yStep</i>	The vertical spacing between grid cells.
<i>yNumCells</i>	The number of grid cell rows.
<i>filter</i>	Type of resampling filter. It can be one of the following: MLIB_NEAREST MLIB_BILINEAR MLIB_BICUBIC MLIB_BICUBIC2
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_SRC_PADDED

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageGridWarp\(3MLIB\)](#), [mllib_ImageGridWarpTable\(3MLIB\)](#),
[mllib_ImageGridWarpTable_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGridWarpTable – grid-based image warp with table-driven interpolation

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>
```

```
mllib_status mllib_ImageGridWarpTable(mllib_image *dst, const mllib_image *src,  
    const mllib_f32 *xWarpPos, const mllib_f32 *yWarpPos,  
    mllib_d64 postShiftX, mllib_d64 postShiftY,  
    mllib_s32 xStart, mllib_s32 xStep, mllib_s32 xNumCells,  
    mllib_s32 yStart, mllib_s32 yStep, mllib_s32 yNumCells,  
    const void *interp_table, mllib_edge edge);
```

Description The `mllib_ImageGridWarpTable()` function performs a regular grid-based image warp with table-driven interpolation. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

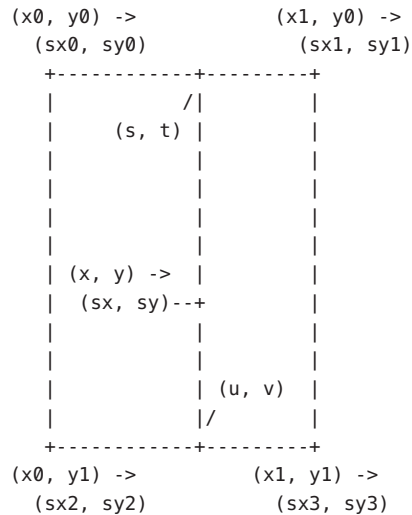
For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

```
xfrac = (x - x0)/(x1 - x0)  
yfrac = (y - y0)/(y1 - y0)  
  
s = sx0 + (sx1 - sx0)*xfrac  
t = sy0 + (sy1 - sy0)*xfrac  
  
u = sx2 + (sx3 - sx2)*xfrac  
v = sy2 + (sy3 - sy2)*xfrac  
  
sx = s + (u - s)*yfrac - postShiftX  
sy = t + (v - t)*yfrac - postShiftY
```

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:



The results of above interpolation are shifted by $(-postShiftX, -postShiftY)$ to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at $(xStart, yStart)$. Each cell has width equal to $xStep$ and height equal to $yStep$, and there are $xNumCells$ cells horizontally and $yNumCells$ cells vertically.

The degree of warping within each cell is defined by the values in $xWarpPos$ and $yWarpPos$ parameters. Each of these parameters must contain $(xNumCells + 1) * (yNumCells + 1)$ values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose $xNumCells$ is equal to 2 and $yNumCells$ is equal to 1. Then the order of the data in the $xWarpPos$ would be:

$x00, x10, x20, x01, x11, x21$

and in the $yWarpPos$:

$y00, y10, y20, y01, y11, y21$

for a total of $(2 + 1) * (1 + 1) = 6$ elements in each table.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

<i>xWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing horizontal warp positions at the grid points, in row-major order.
<i>yWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing vertical warp positions at the grid points, in row-major order.
<i>postShiftX</i>	The displacement to apply to source X positions.
<i>postShiftY</i>	The displacement to apply to source Y positions.
<i>xStart</i>	The minimum X coordinate of the grid.
<i>xStep</i>	The horizontal spacing between grid cells.
<i>xNumCells</i>	The number of grid cell columns.
<i>yStart</i>	The minimum Y coordinate of the grid.
<i>yStep</i>	The vertical spacing between grid cells.
<i>yNumCells</i>	The number of grid cell rows.
<i>interp_table</i>	Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_SRC_PADDED

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInterpTableCreate\(3MLIB\)](#), [mllib_ImageInterpTableDelete\(3MLIB\)](#),
[mllib_ImageGridWarp\(3MLIB\)](#), [mllib_ImageGridWarp_Fp\(3MLIB\)](#),
[mllib_ImageGridWarpTable_Fp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageGridWarpTable_Fp – grid-based image warp with table-driven interpolation

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageGridWarpTable_Fp(mllib_image *dst,
    const mllib_image *src, const mllib_f32 *xWarpPos,
    const mllib_f32 *yWarpPos, mllib_d64 postShiftX, mllib_d64 postShiftY,
    mllib_s32 xStart, mllib_s32 xStep, mllib_s32 xNumCells,
    mllib_s32 yStart, mllib_s32 yStep, mllib_s32 yNumCells,
    const void *interp_table, mllib_edge edge);
```

Description The `mllib_ImageGridWarpTable_Fp()` function performs a regular grid-based image warp on a floating-point image with table-driven interpolation. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at $(0.5, 0.5)$.

For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at $(x0, y0)$, $(x1, y0)$, $(x0, y1)$ and $(x1, y1)$, with source coordinates defined at each respective corner equal to $(sx0, sy0)$, $(sx1, sy1)$, $(sx2, sy2)$ and $(sx3, sy3)$, the source position (sx, sy) that maps onto (x, y) is given by the formulas:

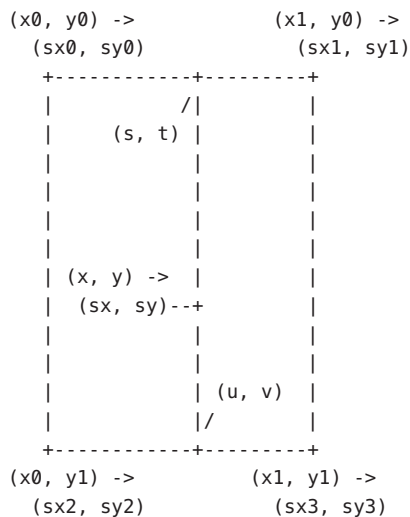
```
xfrac = (x - x0)/(x1 - x0)
yfrac = (y - y0)/(y1 - y0)

s = sx0 + (sx1 - sx0)*xfrac
t = sy0 + (sy1 - sy0)*xfrac

u = sx2 + (sx3 - sx2)*xfrac
v = sy2 + (sy3 - sy2)*xfrac

sx = s + (u - s)*yfrac - postShiftX
sy = t + (v - t)*yfrac - postShiftY
```

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:



The results of above interpolation are shifted by `(-postShiftX, -postShiftY)` to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at `(xStart, yStart)`. Each cell has width equal to `xStep` and height equal to `yStep`, and there are `xNumCells` cells horizontally and `yNumCells` cells vertically.

The degree of warping within each cell is defined by the values in `xWarpPos` and `yWarpPos` parameters. Each of these parameters must contain $(xNumCells + 1) * (yNumCells + 1)$ values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose `xNumCells` is equal to 2 and `yNumCells` is equal to 1. Then the order of the data in the `xWarpPos` would be:

`x00, x10, x20, x01, x11, x21`

and in the `yWarpPos`:

`y00, y10, y20, y01, y11, y21`

for a total of $(2 + 1) * (1 + 1) = 6$ elements in each table.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

<i>xWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing horizontal warp positions at the grid points, in row-major order.
<i>yWarpPos</i>	A float array of length $(xNumCells + 1) * (yNumCells + 1)$ containing vertical warp positions at the grid points, in row-major order.
<i>postShiftX</i>	The displacement to apply to source X positions.
<i>postShiftY</i>	The displacement to apply to source Y positions.
<i>xStart</i>	The minimum X coordinate of the grid.
<i>xStep</i>	The horizontal spacing between grid cells.
<i>xNumCells</i>	The number of grid cell columns.
<i>yStart</i>	The minimum Y coordinate of the grid.
<i>yStep</i>	The vertical spacing between grid cells.
<i>yNumCells</i>	The number of grid cell rows.
<i>interp_table</i>	Pointer to an interpolation table. The table is created by the <code>mlib_ImageInterpTableCreate()</code> function.
<i>edge</i>	Type of edge condition. It can be one of the following: MLIB_EDGE_DST_NO_WRITE MLIB_EDGE_SRC_PADDED

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mlib_ImageInterpTableCreate(3MLIB)`, `mlib_ImageInterpTableDelete(3MLIB)`, `mlib_ImageGridWarp(3MLIB)`, `mlib_ImageGridWarp_Fp(3MLIB)`, `mlib_ImageGridWarpTable(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageHistogram2 – histogram

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageHistogram2(mllib_s32 **histo, const mllib_image *img,  
    const mllib_s32 *numBins, const mllib_s32 *lowValue,  
    const mllib_s32 *highValue, mllib_s32 xStart, mllib_s32 yStart,  
    mllib_s32 xPeriod, mllib_s32 yPeriod);
```

Description The `mllib_ImageHistogram2()` function creates a histogram by scanning an image, counting the number of pixels within a given range for each channel of the image, and then generating a histogram.

The image can have 1, 2, 3 or 4 channels. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The histogram must have the same number of channels as the image has.

One entry of the histogram, or a bin, is used to accumulate the number of pixels within a certain sub-range. The legal pixel range and the number of bins may be controlled separately.

If `binWidth` is defined as $(\text{highValue} - \text{lowValue}) / \text{numBins}$ then bin `i` counts pixel values in the following range:

$$\text{lowValue} + i * \text{binWidth} \leq x < \text{lowValue} + (i + 1) * \text{binWidth}$$

The set of pixels scanned may furthermore be reduced by specifying `xPeriod` and `yPeriod` parameters that specify the sampling rate along each axis.

The set of pixels to be accumulated may be obtained from the following equation:

$$\begin{aligned} x &= xStart + p * xPeriod; & 0 \leq p < (w - xStart) / xPeriod \\ y &= yStart + q * yPeriod; & 0 \leq q < (h - yStart) / yPeriod \end{aligned}$$

It is the user's responsibility to clear the histogram table before this function is called and to ensure that the histogram table supplied is suitable for the source image and the parameters. Otherwise, the result of this function is undefined.

The range from `lowValue[k]` to `(highValue[k] - 1)` must be a valid subrange of the image type range.

Parameters The function takes the following arguments:

<i>histo</i>	Pointer to histogram. The format of the histogram is <code>histo[channel][index]</code> . The index values for channel <code>i</code> can be <code>0, 1, ..., numBins[i]-1</code> .
<i>img</i>	Pointer to source image.
<i>numBins</i>	The number of bins for each channel of the image.
<i>lowValue</i>	The lowest pixel value checked for each channel.

- highValue* The highest pixel value checked for each channel. When counting the pixel values, highValue is not included.
- xStart* The initial X sample coordinate.
- yStart* The initial Y sample coordinate.
- xPeriod* The X sampling rate. xPeriod ≥ 1.
- yPeriod* The Y sampling rate. yPeriod ≥ 1.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageHistogram\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageHistogram – histogram

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageHistogram(mllib_s32 **histo, const mllib_image *img);`

Description The `mllib_ImageHistogram()` function creates a histogram. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

Parameters The function takes the following arguments:

histo Pointer to histogram. The format of the histogram is `histo[channel][index]`. The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768 to -1, then from 0 to 32767. The `MLIB_USHORT` type entries are indexed from 0 to 65535. The `MLIB_INT` type entries are indexed from -2147483648 to -1, then from 0 to 2147483647.

img Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageHistogram2\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInterpTableCreate – creates an interpolation table

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

```
void *mllib_ImageInterpTableCreate(mllib_type type, mllib_s32 width,
    mllib_s32 height, mllib_s32 leftPadding, mllib_s32 topPadding,
    mllib_s32 subsampleBitsH, mllib_s32 subsampleBitsV,
    mllib_s32 precisionBits, const void *dataH, const void *dataV);
```

Description The `mllib_ImageInterpTableCreate()` function creates an interpolation table based on parameters specified.

This function creates an internal data structure, an interpolation table, which can be used by some image geometric functions for implementing a table-driven interpolation algorithm.

The parameter `type` defines the type of `dataH/dataV` input arrays and can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The `dataH` array should have at least `width*2**subsampleBitsH` entries. `dataH[i*2**subsampleBitsH]` holds the coefficient for the leftmost neighboring pixel, `dataH[i*2**subsampleBitsH + 1]` holds the coefficient for the second neighboring pixel from left, ..., and `dataH[i*2**subsampleBitsH + width - 1]` holds the coefficient for the rightmost neighboring pixel, where `i = 0, 1, 2, ..., 2**subsampleBitsH - 1`.

The `dataV` array should have at least `height*2**subsampleBitsV` entries or should be `NULL`. If `dataV` is `NULL`, then `dataH` is used in its place, and in this case the parameters `topPadding`, `height`, and `subsampleBitsV` are ignored.

Parameters The function takes the following arguments:

<i>type</i>	Data type of the coefficients.
<i>width</i>	Width of the interpolation kernel in pixels.
<i>height</i>	Height of the interpolation kernel in pixels.
<i>leftPadding</i>	Number of pixels lying to the left of the interpolation kernel key position.
<i>topPadding</i>	Number of pixels lying above the interpolation kernel key position.
<i>subsampleBitsH</i>	Numbers of bits used for the horizontal subsample position.
<i>subsampleBitsV</i>	Numbers of bits used for the vertical subsample position.
<i>precisionBits</i>	Number of fractional bits used to describe the coefficients.
<i>dataH</i>	Pointer to horizontal coefficient data.
<i>dataV</i>	Pointer to vertical coefficient data.

Return Values The function returns a pointer to an interpolation table.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInterpTableDelete\(3MLIB\)](#), [mllib_ImageAffineTable\(3MLIB\)](#), [mllib_ImageZoomTranslateTable\(3MLIB\)](#), [mllib_ImageGridWarpTable\(3MLIB\)](#), [mllib_ImagePolynomialWarpTable\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInterpTableDelete – deletes an interpolation table

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>
```

```
void mllib_ImageInterpTableDelete(void *interp_table);
```

Description The mllib_ImageInterpTableDelete() function deletes an interpolation table.

This function deletes the structure of an interpolation table and frees the memory allocated by mllib_ImageInterpTableCreate().

Parameters The function takes the following arguments:

interp_table Pointer to an interpolation table.

Return Values None.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInterpTableCreate\(3MLIB\)](#), [mllib_ImageAffineTable\(3MLIB\)](#),
[mllib_ImageZoomTranslateTable\(3MLIB\)](#), [mllib_ImageGridWarpTable\(3MLIB\)](#),
[mllib_ImagePolynomialWarpTable\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInvert – invert

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

`mllib_status mllib_ImageInvert(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageInvert()` function performs the inversion of an image such that white becomes black, light gray becomes dark gray, and so on.

It uses the following equation:

$$dst[x][y][i] = (Gwhite + Gblack) - src[x][y][i]$$

The values of `Gwhite` and `Gblack` for different types of images are:

Image Type	Gwhite	Gblack	Gwhite + Gblack
MLIB_BYTE	255	0	255 (0xFF)
MLIB_SHORT	32767	-32768	-1 (0xFFFF)
MLIB_USHORT	65535	0	65535 (0xFFFF)
MLIB_INT	2147483647	-2147483648	-1 (0xFFFFFFFF)

Given that integer data are in the two's complement representation, `mllib_ImageInvert()` is the same as `mllib_ImageNot()`, while `mllib_ImageInvert_Inp()` is the same as `mllib_ImageNot_Inp()`.

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInvert_Inp\(3MLIB\)](#), [mllib_ImageInvert_Fp\(3MLIB\)](#), [mllib_ImageInvert_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInvert_Fp – invert

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageInvert_Fp(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageInvert_Fp()` function performs the floating-point inversion of an image such that white becomes black, light gray becomes dark gray, and so on.

It uses the following equation:

$$dst[x][y][i] = -src[x][y][i]$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTETYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInvert_Fp_Inp\(3MLIB\)](#), [mllib_ImageInvert\(3MLIB\)](#),
[mllib_ImageInvert_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInvert_Fp_Inp – invert

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageInvert_Fp_Inp(mllib_image *srcdst);`

Description The `mllib_ImageInvert_Fp_Inp()` function performs the floating-point inversion of an image such that white becomes black, light gray becomes dark gray, and so on, in place.

It uses the following equation:

`srcdst[x][y][i] = -srcdst[x][y][i]`

Parameters The function takes the following arguments:

`srcdst` Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInvert_Fp\(3MLIB\)](#), [mllib_ImageInvert\(3MLIB\)](#),
[mllib_ImageInvert_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageInvert_Inp – invert in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageInvert_Inp(mllib_image *srcdst);
```

Description The `mllib_ImageInvert_Inp()` function performs the in-place inversion of an image such that white becomes black, light gray becomes dark gray, and so on.

It uses the following equation:

$$\text{srcdst}[x][y][i] = (\text{Gwhite} + \text{Gblack}) - \text{srcdst}[x][y][i]$$

The values of `Gwhite` and `Gblack` for different types of images are:

Image Type	Gwhite	Gblack	Gwhite + Gblack
MLIB_BYTE	255	0	255 (0xFF)
MLIB_SHORT	32767	-32768	-1 (0xFFFF)
MLIB_USHORT	65535	0	65535 (0xFFFF)
MLIB_INT	2147483647	-2147483648	-1 (0xFFFFFFFF)

Given that integer data are in the two's complement representation, `mllib_ImageInvert()` is the same as `mllib_ImageNot()`, while `mllib_ImageInvert_Inp()` is the same as `mllib_ImageNot_Inp()`.

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageInvert\(3MLIB\)](#), [mllib_ImageInvert_Fp\(3MLIB\)](#), [mllib_ImageInvert_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageIsNotAligned2 – image query, two-byte aligned

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotAligned2(const mllib_image *img);`

Description The `mllib_ImageIsNotAligned2()` function tests for a specific alignment of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if data address is two-byte aligned; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageIsNotAligned4\(3MLIB\)](#), [mllib_ImageIsNotAligned8\(3MLIB\)](#), [mllib_ImageIsNotAligned64\(3MLIB\)](#), [mllib_ImageIsNotHeight2X\(3MLIB\)](#), [mllib_ImageIsNotHeight4X\(3MLIB\)](#), [mllib_ImageIsNotHeight8X\(3MLIB\)](#), [mllib_ImageIsNotOneDvector\(3MLIB\)](#), [mllib_ImageIsNotStride8X\(3MLIB\)](#), [mllib_ImageIsNotWidth2X\(3MLIB\)](#), [mllib_ImageIsNotWidth4X\(3MLIB\)](#), [mllib_ImageIsNotWidth8X\(3MLIB\)](#), [mllib_ImageIsUserAllocated\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageIsNotAligned4 – image query, four-byte aligned

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
int mllib_ImageIsNotAligned4(const mllib_image *img);
```

Description The `mllib_ImageIsNotAligned4()` function tests for a specific alignment of a mediaLib image structure.

Parameters The function takes the following arguments:
img Pointer to source image.

Return Values Returns 0 if data address is four-byte aligned; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`, `mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`, `mllib_ImageIsNotHeight8X(3MLIB)`, `mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`, `mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`, `mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotAligned64 – image query, 64-byte aligned

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
int mllib_ImageIsNotAligned64(const mllib_image *img);
```

Description The `mllib_ImageIsNotAligned64()` function tests for a specific alignment characteristic of a `mediaLib` image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if data address is 64-byte aligned; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageIsNotAligned2\(3MLIB\)](#), [mllib_ImageIsNotAligned4\(3MLIB\)](#),
[mllib_ImageIsNotAligned8\(3MLIB\)](#), [mllib_ImageIsNotHeight2X\(3MLIB\)](#),
[mllib_ImageIsNotHeight4X\(3MLIB\)](#), [mllib_ImageIsNotHeight8X\(3MLIB\)](#),
[mllib_ImageIsNotOneDvector\(3MLIB\)](#), [mllib_ImageIsNotStride8X\(3MLIB\)](#),
[mllib_ImageIsNotWidth2X\(3MLIB\)](#), [mllib_ImageIsNotWidth4X\(3MLIB\)](#),
[mllib_ImageIsNotWidth8X\(3MLIB\)](#), [mllib_ImageIsUserAllocated\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageIsNotAligned8 – image query, eight-byte aligned

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotAligned8(const mllib_image *img);`

Description The `mllib_ImageIsNotAligned8()` function tests for a specific alignment of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if data address is eight-byte aligned; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned64(3MLIB)`, `mllib_ImageIsNotHeight2X(3MLIB)`,
`mllib_ImageIsNotHeight4X(3MLIB)`, `mllib_ImageIsNotHeight8X(3MLIB)`,
`mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`,
`mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotHeight2X – image query, 2X height

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotHeight2X(const mllib_image *img);`

Description The `mllib_ImageIsNotHeight2X()` function tests for a specific height characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if height is a multiple of two; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight4X(3MLIB)`, `mllib_ImageIsNotHeight8X(3MLIB)`,
`mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`,
`mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotHeight4X – image query, 4X height

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotHeight4X(const mllib_image *img);`

Description The `mllib_ImageIsNotHeight4X()` function tests for a specific height characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if height is a multiple of four; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight8X(3MLIB)`,
`mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`,
`mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotHeight8X – image query, 8X height

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotHeight8X(const mllib_image *img);`

Description The `mllib_ImageIsNotHeight8X()` function tests for a specific height characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if height is a multiple of eight; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`,
`mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`,
`mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotOneDvector – image query, 1D vector

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotOneDvector(const mllib_image *img);`

Description The `mllib_ImageIsNotOneDvector()` function tests for a specific dimension characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if data space can be treated as a 1D vector; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`,
`mllib_ImageIsNotHeight8X(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`,
`mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotStride8X – image query, 8X stride

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotStride8X(const mllib_image *img);`

Description The `mllib_ImageIsNotStride8X()` function tests for a specific stride characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if stride is a multiple of eight; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageIsNotAligned2\(3MLIB\)](#), [mllib_ImageIsNotAligned4\(3MLIB\)](#),
[mllib_ImageIsNotAligned8\(3MLIB\)](#), [mllib_ImageIsNotAligned64\(3MLIB\)](#),
[mllib_ImageIsNotHeight2X\(3MLIB\)](#), [mllib_ImageIsNotHeight4X\(3MLIB\)](#),
[mllib_ImageIsNotHeight8X\(3MLIB\)](#), [mllib_ImageIsNotOneDvector\(3MLIB\)](#),
[mllib_ImageIsNotWidth2X\(3MLIB\)](#), [mllib_ImageIsNotWidth4X\(3MLIB\)](#),
[mllib_ImageIsNotWidth8X\(3MLIB\)](#), [mllib_ImageIsUserAllocated\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageIsNotWidth2X – image query, 2X width

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsNotWidth2X(const mllib_image *img);`

Description The `mllib_ImageIsNotWidth2X()` function tests for a specific width characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if width is a multiple of two; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`,
`mllib_ImageIsNotHeight8X(3MLIB)`, `mllib_ImageIsNotOneDvector(3MLIB)`,
`mllib_ImageIsNotStride8X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`,
`mllib_ImageIsNotWidth8X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsNotWidth4X – image query, 4X width

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
int mllib_ImageIsNotWidth4X(const mllib_image *img);
```

Description The `mllib_ImageIsNotWidth4X()` function tests for a specific width characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

img Pointer to source image.

Return Values Returns 0 if width is a multiple of four; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageIsNotAligned2\(3MLIB\)](#), [mllib_ImageIsNotAligned4\(3MLIB\)](#),
[mllib_ImageIsNotAligned8\(3MLIB\)](#), [mllib_ImageIsNotAligned64\(3MLIB\)](#),
[mllib_ImageIsNotHeight2X\(3MLIB\)](#), [mllib_ImageIsNotHeight4X\(3MLIB\)](#),
[mllib_ImageIsNotHeight8X\(3MLIB\)](#), [mllib_ImageIsNotOneDvector\(3MLIB\)](#),
[mllib_ImageIsNotStride8X\(3MLIB\)](#), [mllib_ImageIsNotWidth2X\(3MLIB\)](#),
[mllib_ImageIsNotWidth8X\(3MLIB\)](#), [mllib_ImageIsUserAllocated\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageIsNotWidth8X – image query, 8X width

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
int mllib_ImageIsNotWidth8X(const mllib_image *img);
```

Description The `mllib_ImageIsNotWidth8X()` function tests for a specific width characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:
img Pointer to source image.

Return Values Returns 0 if width is a multiple of eight; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`, `mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`, `mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`, `mllib_ImageIsNotHeight8X(3MLIB)`, `mllib_ImageIsNotOneDvector(3MLIB)`, `mllib_ImageIsNotStride8X(3MLIB)`, `mllib_ImageIsNotWidth2X(3MLIB)`, `mllib_ImageIsNotWidth4X(3MLIB)`, `mllib_ImageIsUserAllocated(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageIsUserAllocated – image query, user-allocated

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`int mllib_ImageIsUserAllocated(const mllib_image *src);`

Description The `mllib_ImageIsUserAllocated()` function tests for a specific allocation characteristic of a `mediaLib` image structure.

Parameters The function takes the following arguments:

`src` Pointer to source image.

Return Values Returns 0 if data space has been allocated by `mediaLib`; otherwise, returns nonzero.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `mllib_ImageIsNotAligned2(3MLIB)`, `mllib_ImageIsNotAligned4(3MLIB)`,
`mllib_ImageIsNotAligned8(3MLIB)`, `mllib_ImageIsNotAligned64(3MLIB)`,
`mllib_ImageIsNotHeight2X(3MLIB)`, `mllib_ImageIsNotHeight4X(3MLIB)`,
`mllib_ImageIsNotHeight8X(3MLIB)`, `mllib_ImageIsNotOneDvector(3MLIB)`,
`mllib_ImageIsNotStride8X(3MLIB)`, `mllib_ImageIsNotWidth2X(3MLIB)`,
`mllib_ImageIsNotWidth4X(3MLIB)`, `mllib_ImageIsNotWidth8X(3MLIB)`, [attributes\(5\)](#)

Name mllib_ImageLog – computes the natural logarithm of the image pixels

Synopsis `cc [flag...] file... -lmllib [library...]`
`#include <mllib.h>`

`mllib_status mllib_ImageLog(mllib_image *dst, const mllib_image *src);`

Description The `mllib_ImageLog()` function computes the natural logarithm of the image pixels.

It uses the following equation:

$dst[x][y][i] = \log(src[x][y][i])$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLog_Fp\(3MLIB\)](#), [mllib_ImageLog_Fp_Inp\(3MLIB\)](#), [mllib_ImageLog_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLog_Fp – computes the natural logarithm of the image pixels

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_ImageLog_Fp(mllib_image *dst, const mllib_image *src);
```

Description The mllib_ImageLog_Fp() function computes the natural logarithm of the image pixels.

It uses the following equation:

$$dst[x][y][i] = \log(src[x][y][i])$$

Parameters The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLog\(3MLIB\)](#), [mllib_ImageLog_Fp_Inp\(3MLIB\)](#), [mllib_ImageLog_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLog_Fp_Inp – computes the natural logarithm of the image pixels

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>
```

```
mllib_status mllib_ImageLog_Fp_Inp(mllib_image *srcdst);
```

Description The `mllib_ImageLog_Fp_Inp()` function computes the natural logarithm of the image pixels, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \log(\text{srcdst}[x][y][i])$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLog\(3MLIB\)](#), [mllib_ImageLog_Fp\(3MLIB\)](#), [mllib_ImageLog_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLog_Inp – computes the natural logarithm of the image pixels, in place

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mllib.h>

mllib_status mllib_ImageLog_Inp(mllib_image *srcdst);
```

Description The `mllib_ImageLog_Inp()` function computes the natural logarithm of the image pixels in place.

It uses the following equation:

$$srcdst[x][y][i] = \log(srcdst[x][y][i])$$

Parameters The function takes the following arguments:

srcdst Pointer to source and destination image.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLog\(3MLIB\)](#), [mllib_ImageLog_Fp\(3MLIB\)](#), [mllib_ImageLog_Fp_Inp\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLookUp2 – table lookup

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageLookUp2(mllib_image *dst, const mllib_image *src,  
    const void **table, const mllib_s32 *offsets, mllib_s32 channels);
```

Description The `mllib_ImageLookUp2()` function maps the source image to the destination image by using the user-specified lookup table and an offset.

The source and destination images must have the same width and height.

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

Type[*]	BYTE	SHORT	USHORT	INT	FLOAT	DOUBLE
MLIB_BIT	Y					
MLIB_BYTE	Y	Y	Y	Y	Y	Y
MLIB_SHORT	Y	Y	Y	Y	Y	Y
MLIB_USHORT	Y	Y	Y	Y	Y	Y
MLIB_INT	Y	Y	Y	Y	Y	Y

[*] Each row represents a source data type. Each column represents a destination data type.

The source and destination images also can have a different number of channels. The source image can be a single-channel image or can have the same number of channels as the destination image. The lookup table can have one channel or have the same channels as the destination image. See the following table for possible variations on the number of channels in the images and the lookup table:

# of channels in the input image	# of channels in the lookup table	# of channels in the output image
1	n	n
n	1	n
n	n	n

where, $n = 1, 2, 3, 4$.

Each of the following equations is used in the corresponding case shown in the table above.

```
dst[x][y][i] = table[i][src[x][y][0] - offsets[i]]
dst[x][y][i] = table[0][src[x][y][i] - offsets[0]]
dst[x][y][i] = table[i][src[x][y][i] - offsets[i]]
```

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- table* Pointer to lookup table. The data type of the lookup table is the same as that of the destination image.. The format of the lookup table is:

table[channel][index]

The entries are indexed from 0 to 1, 2, ..., and so on. It is the user's responsibility to provide a lookup table that has enough entries to cover all possible values of the pixel components deducted by the offset in each channel of the source image.
- offsets* Offset values subtracted from the src pixel before table lookup.
- channels* Number of channels in the lookup table. If the number of channels equals 1, then the same table is applied to all channels. Otherwise, the number of channels must be no less than the number of channels in the destination image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLookUp\(3MLIB\)](#), [mllib_ImageLookUp_Inp\(3MLIB\)](#), [mllib_ImageLookUpMask\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLookUp – table lookup

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageLookUp(mllib_image *dst, const mllib_image *src,  
                               const void **table);
```

Description The `mllib_ImageLookUp()` function maps the source image to the destination image by using the user-specified lookup table.

The source and destination images must have the same width and height. The source image can be a single channel image or can have the same number of channels as the destination image. One of the following equations is used accordingly:

```
dst[x][y][i] = table[i][src[x][y][0]]  
dst[x][y][i] = table[i][src[x][y][i]]
```

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

Type[*]	BYTE	SHORT	USHORT	INT	FLOAT	DOUBLE
MLIB_BIT	Y					
MLIB_BYTE	Y	Y	Y	Y	Y	Y
MLIB_SHORT	Y	Y	Y	Y	Y	Y
MLIB_USHORT	Y	Y	Y	Y	Y	Y
MLIB_INT	Y	Y	Y	Y	Y	Y

[*] Each row represents a source data type. Each column represents a destination data type.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- table* Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:

```
table[channel][index]
```

The `MLIB_BIT` type entries are indexed from 0 to 1. The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768 to -1,

then from 0 to 32767. The MLIB_USHORT type entries are indexed from 0 to 65535. The MLIB_INT type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

If a table covering the full range of input data type is not available or not realistic, which is mostly true for doing table lookup with an MLIB_INT input image, a smaller table can be used. In this case, the pointer to the table has to be adjusted as if it is pointing to the element for the smallest value of the input data type. For example, to use a table covering input data range of [-65536 , 65535], the pointer needs to be adjusted as follows:

```
table_16_32[0] += MLIB_S32_MIN + 65536;
```

This might cause a pointer arithmetic overflow in 32-bit mode, but probably works if the overflow is handled as a wrap-around. If possible, function `mllib_ImageLookup2()` should be used instead.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mllib_ImageLookup_Inp\(3MLIB\)](#), [mllib_ImageLookup2\(3MLIB\)](#), [mllib_ImageLookupMask\(3MLIB\)](#), [attributes\(5\)](#)

Name mlib_ImageLookUp_Inp – table lookup, in place

Synopsis `cc [flag...] file... -lmlib [library...]
#include <mlib.h>`

```
mlib_status mlib_ImageLookUp_Inp(mlib_image *srcdst, const void **table);
```

Description The `mlib_ImageLookUp_Inp()` function maps the source image to the destination image, in place, by using the user-specified lookup table.

The following equation is used:

$$\text{srcdst}[x][y][i] = \text{table}[i][\text{srcdst}[x][y][i]]$$

Parameters The function takes the following arguments:

srcdst Pointer to first source and destination image.

table Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:

```
table[channel][index]
```

The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768 to -1, then from 0 to 32767. The `MLIB_USHORT` type entries are indexed from 0 to 65535. The `MLIB_INT` type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See [attributes\(5\)](#) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also [mlib_ImageLookUp\(3MLIB\)](#), [mlib_ImageLookUp2\(3MLIB\)](#), [mlib_ImageLookUpMask\(3MLIB\)](#), [attributes\(5\)](#)

Name mllib_ImageLookupMask – table lookup with mask

Synopsis `cc [flag...] file... -lmllib [library...]
#include <mllib.h>`

```
mllib_status mllib_ImageLookupMask(mllib_image *dst, const mllib_image *src,  
    const void **table, mllib_s32 channels, mllib_s32 cmask);
```

Description The `mllib_ImageLookupMask()` function maps the source image to the destination image by using the user-specified lookup table and applying a channel mask.

The source and destination images must have the same width and height. The source image can be a single channel image or can have the same number of channels as the destination image. One of the following equations is used accordingly:

```
dst[x][y][i] = table[i][src[x][y][0]]  
dst[x][y][i] = table[i][src[x][y][i]]
```

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

Type[*]	BYTE	SHORT	USHORT	INT	FLOAT	DOUBLE
MLIB_BIT	Y					
MLIB_BYTE	Y	Y	Y	Y	Y	Y
MLIB_SHORT	Y	Y	Y	Y	Y	Y
MLIB_USHORT	Y	Y	Y	Y	Y	Y
MLIB_INT	Y	Y	Y	Y	Y	Y

[*] Each row represents a source data type. Each column represents a destination data type.

Parameters The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- table* Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:
- ```
table[channel][index]
```
- The `MLIB_BIT` type entries are indexed from 0 to 1. The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768



to -1, then from 0 to 32767. The MLIB\_USHORT type entries are indexed from 0 to 65535. The MLIB\_INT type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

- channels* Number of channels in the lookup table. If the number of channels is equal to 1, then the same table is applied to all channels. Otherwise, the number of channels must be no less than the number of valid 1s in the channel mask.
- cmask* Channel mask. Each bit of the mask represents a channel of an image or a lookup table. Only the rightmost four bits of cmask are considered, where the least significant bit of cmask is for the last channel. The channels corresponding to 0 bits of cmask are not processed or used. cmask is always applied to the destination image dst. If the source image src has the same number of channels as dst, then cmask is also applied to src. Otherwise, each channel of src is used for each cmask bit with a value of 1, in this order: the first channel for the first 1 from the left in cmask. If src has only one channel, then the same src channel is used for every cmask bit with a value of 1. If the lookup table has the same number of channels as dst, then cmask is also applied to table. Otherwise, each table channel is used for each cmask bit with a value of 1, in this order: the first channel for the first 1 from the left in cmask. If table has only one channel, then the same table channel is used for every cmask bit with a value of 1.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageLookUp\(3MLIB\)](#), [mllib\\_ImageLookUp\\_Inp\(3MLIB\)](#), [mllib\\_ImageLookUp2\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMax – two image maximum

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMax(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMax()` function accepts input from the two source images and writes the maximum to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \text{MAX}\{ src1[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src1*     Pointer to first source image.
- src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMax\\_Fp\(3MLIB\)](#), [mllib\\_ImageMax\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageMax\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMaxFilter3x3 – 3x3 maximum filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMaxFilter3x3(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMaxFilter3x3()` function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where  $x = 1, \dots, w - 2$ ;  $y = 1, \dots, h - 2$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3_Fp(3MLIB)`, `mllib_ImageMaxFilter5x5(3MLIB)`,  
`mllib_ImageMaxFilter5x5_Fp(3MLIB)`, `mllib_ImageMaxFilter7x7(3MLIB)`,  
`mllib_ImageMaxFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3(3MLIB)`,  
`mllib_ImageMedianFilter3x3_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3_US(3MLIB)`,  
`mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

Name

mllib\_ImageMaxFilter3x3\_Fp – 3x3 maximum filter

Synopsis

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMaxFilter3x3_Fp(mllib_image *dst,
 const mllib_image *src);
```

Description

The `mllib_ImageMaxFilter3x3_Fp()` function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$dst[x][y][0] = \text{MAX}\{ src[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where  $x = 1, \dots, w - 2$ ;  $y = 1, \dots, h - 2$ .

Parameters

The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

See Also

[mllib\\_ImageMaxFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#),

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMaxFilter5x5 – 5x5 Max Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMaxFilter5x5(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMaxFilter5x5()` function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \\ x-2 \leq p \leq x+2; y-2 \leq q \leq y+2 \}$$

where  $x = 2, \dots, w - 3$ ;  $y = 2, \dots, h - 3$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5_Fp(3MLIB)`, `mllib_ImageMaxFilter7x7(3MLIB)`,  
`mllib_ImageMaxFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3(3MLIB)`,  
`mllib_ImageMedianFilter3x3_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3_US(3MLIB)`,  
`mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```



**Name** mllib\_ImageMaxFilter5x5\_Fp – 5x5 Max Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMaxFilter5x5_Fp(mllib_image *dst,
 const mllib_image *src);
```

**Description** The `mllib_ImageMaxFilter5x5_Fp()` function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \\ x-2 \leq p \leq x+2; y-2 \leq q \leq y+2 \}$$

where  $x = 2, \dots, w - 3$ ;  $y = 2, \dots, h - 3$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter7x7(3MLIB)`,  
`mllib_ImageMaxFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3(3MLIB)`,  
`mllib_ImageMedianFilter3x3_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3_US(3MLIB)`,  
`mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMaxFilter7x7 – 7x7 Max Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMaxFilter7x7(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMaxFilter7x7()` function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \\ x-3 \leq p \leq x+3; y-3 \leq q \leq y+3 \}$$

where  $x = 3, \dots, w - 4$ ;  $y = 3, \dots, h - 4$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3(3MLIB)`,  
`mllib_ImageMedianFilter3x3_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3_US(3MLIB)`,  
`mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name**

mllib\_ImageMaxFilter7x7\_Fp – 7x7 Max Filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMaxFilter7x7_Fp(mllib_image *dst,
 const mllib_image *src);
```

**Description**

The `mllib_ImageMaxFilter7x7_Fp()` function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$dst[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \\ x-3 \leq p \leq x+3; y-3 \leq q \leq y+3 \}$$

where  $x = 3, \dots, w - 4$ ;  $y = 3, \dots, h - 4$ .

**Parameters**

The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**

[mllib\\_ImageMaxFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#),
[mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#),
[mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#),
[mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#),

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mlib\_ImageMax\_Fp – two image maximum

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_ImageMax_Fp(mlib_image *dst, const mlib_image *src1,
 const mlib_image *src2);
```

**Description** The `mlib_ImageMax_Fp()` function accepts input from the two floating-point source images and writes the maximum to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \text{MAX}\{ src1[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageMax\(3MLIB\)](#), [mlib\\_ImageMax\\_Fp\\_Inp\(3MLIB\)](#), [mlib\\_ImageMax\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMax\_Fp\_Inp – two image maximum

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMax_Fp_Inp(mllib_image *src1dst, const mllib_image *src2);
```

**Description** The `mllib_ImageMax_Fp_Inp()` function accepts input from the two floating-point source images and writes the maximum, in place, on a pixel-by-pixel basis.

It uses the following equation:

```
src1dst[x][y][i] = MAX{ src1dst[x][y][i], src2[x][y][i] }
```

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMax\(3MLIB\)](#), [mllib\\_ImageMax\\_Fp\(3MLIB\)](#), [mllib\\_ImageMax\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageMaximum – image maximum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMaximum(mllib_s32 *max, const mllib_image *img);`

**Description** The `mllib_ImageMaximum()` function determines the maximum value for each channel in an image.

It uses the following equation:

$$\text{max}[i] = \text{MAX}\{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \}$$

**Parameters** The function takes the following arguments:

*max*      Pointer to maximum vector, where length is the number of channels in the image.  
          `max[i]` contains the maximum of channel `i`.

*img*      Pointer to a source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaximum\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinimum\(3MLIB\)](#),  
[mllib\\_ImageMinimum\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMaximum\_Fp – image maximum

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMaximum_Fp(mllib_d64 *max, const mllib_image *img);
```

**Description** The `mllib_ImageMaximum_Fp()` function determines the maximum value for each channel in a floating-point image.

It uses the following equation:

$$\text{max}[i] = \text{MAX}\{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \}$$

**Parameters** The function takes the following arguments:

*max*      Pointer to maximum vector, where length is the number of channels in the image. `max[i]` contains the maximum of channel `i`.

*img*      Pointer to a source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaximum\(3MLIB\)](#), [mllib\\_ImageMinimum\(3MLIB\)](#), [mllib\\_ImageMinimum\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMax\_Inp – two image maximum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageMax_Inp(mllib_image *src1dst, const mllib_image *src2);
```

**Description** The `mllib_ImageMax_Inp()` function accepts input from the two source images and writes the maximum in place on a pixel-by-pixel basis.

It uses the following equation:

$$src1dst[x][y][i] = \text{MAX}\{ src1dst[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMax\(3MLIB\)](#), [mllib\\_ImageMax\\_Fp\(3MLIB\)](#), [mllib\\_ImageMax\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMean – image mean

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMean(mllib_d64 *mean, const mllib_image *img);`

**Description** The `mllib_ImageMean()` function computes the mean value of all the pixels in the image.

It uses the following equation:

$$\text{mean}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]$$

**Parameters** The function takes the following arguments:

*mean*     Pointer to mean array, where length is the number of channels in the image. `mean[i]` contains the mean of channel `i`.

*img*       Pointer to an image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMean\\_Fp\(3MLIB\)](#), [mllib\\_ImageStdDev\(3MLIB\)](#), [mllib\\_ImageStdDev\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMean\_Fp – image mean

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMean_Fp(mllib_d64 *mean, const mllib_image *img);`

**Description** The `mllib_ImageMean_Fp()` function computes the mean value of all the pixels in the image.

It uses the following equation:

$$\text{mean}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]$$

**Parameters** The function takes the following arguments:

*mean*     Pointer to mean array, where length is the number of channels in the image. `mean[i]` contains the mean of channel `i`.

*img*       Pointer to an image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMean\(3MLIB\)](#), [mllib\\_ImageStdDev\(3MLIB\)](#), [mllib\\_ImageStdDev\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMedianFilter3x3 – 3x3 median filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMedianFilter3x3(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter3x3()` function performs median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
`MLIB_MEDIAN_MASK_RECT`  
`MLIB_MEDIAN_MASK_PLUS`  
`MLIB_MEDIAN_MASK_X`  
`MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
`MLIB_EDGE_DST_NO_WRITE`  
`MLIB_EDGE_DST_FILL_ZERO`  
`MLIB_EDGE_DST_COPY_SRC`  
`MLIB_EDGE_SRC_EXTEND`

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#),

---

```
mllib_ImageMedianFilter5x5_US(3MLIB),mllib_ImageMedianFilter7x7(3MLIB),
mllib_ImageMedianFilter7x7_Fp(3MLIB),mllib_ImageMedianFilter7x7_US(3MLIB),
mllib_ImageMedianFilterMxN(3MLIB),mllib_ImageMedianFilterMxN_Fp(3MLIB),
mllib_ImageMedianFilterMxN_US(3MLIB),mllib_ImageMinFilter3x3(3MLIB),
mllib_ImageMinFilter3x3_Fp(3MLIB),mllib_ImageMinFilter5x5(3MLIB),
mllib_ImageMinFilter5x5_Fp(3MLIB),mllib_ImageMinFilter7x7(3MLIB),
mllib_ImageMinFilter7x7_Fp(3MLIB),mllib_ImageRankFilter3x3(3MLIB),
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMedianFilter3x3\_Fp – 3x3 median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter3x3_Fp(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter3x3_Fp()` function performs floating-point median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to source image.
- mmask*          Shape of the mask to be used for median filtering. It can be one of the following:
- ```
MLIB_MEDIAN_MASK_RECT
MLIB_MEDIAN_MASK_PLUS
MLIB_MEDIAN_MASK_X
MLIB_MEDIAN_MASK_RECT_SEPARABLE
```
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:
- ```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND
```

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#),



```
mllib_ImageMedianFilter5x5(3MLIB),mllib_ImageMedianFilter5x5_Fp(3MLIB),
mllib_ImageMedianFilter5x5_US(3MLIB),mllib_ImageMedianFilter7x7(3MLIB),
mllib_ImageMedianFilter7x7_Fp(3MLIB),mllib_ImageMedianFilter7x7_US(3MLIB),
mllib_ImageMedianFilterMxN(3MLIB),mllib_ImageMedianFilterMxN_Fp(3MLIB),
mllib_ImageMedianFilterMxN_US(3MLIB),mllib_ImageMinFilter3x3(3MLIB),
mllib_ImageMinFilter3x3_Fp(3MLIB),mllib_ImageMinFilter5x5(3MLIB),
mllib_ImageMinFilter5x5_Fp(3MLIB),mllib_ImageMinFilter7x7(3MLIB),
mllib_ImageMinFilter7x7_Fp(3MLIB),mllib_ImageRankFilter3x3(3MLIB),
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMedianFilter3x3\_US – 3x3 median filter, unsigned

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter3x3_US(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge, mllib_s32 bits);
```

**Description** The `mllib_ImageMedianFilter3x3_US()` function performs median filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
`MLIB_MEDIAN_MASK_RECT`  
`MLIB_MEDIAN_MASK_PLUS`  
`MLIB_MEDIAN_MASK_X`  
`MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
`MLIB_EDGE_DST_NO_WRITE`  
`MLIB_EDGE_DST_FILL_ZERO`  
`MLIB_EDGE_DST_COPY_SRC`  
`MLIB_EDGE_SRC_EXTEND`
- bits* The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,  
`mllib_ImageRankFilter5x5_US(3MLIB)`, `mllib_ImageRankFilter7x7(3MLIB)`,  
`mllib_ImageRankFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter7x7_US(3MLIB)`,  
`mllib_ImageRankFilterMxN(3MLIB)`, `mllib_ImageRankFilterMxN_Fp(3MLIB)`,  
`mllib_ImageRankFilterMxN_US(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMedianFilter5x5 – 5x5 median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter5x5(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter5x5()` function performs median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#),

---

```
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageMedianFilter5x5\_Fp – 5x5 median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter5x5_Fp(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter5x5_Fp()` function performs floating-point median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),

```
mllib_ImageMedianFilter3x3_US(3MLIB),mllib_ImageMedianFilter5x5(3MLIB),
mllib_ImageMedianFilter5x5_US(3MLIB),mllib_ImageMedianFilter7x7(3MLIB),
mllib_ImageMedianFilter7x7_Fp(3MLIB),mllib_ImageMedianFilter7x7_US(3MLIB),
mllib_ImageMedianFilterMxN(3MLIB),mllib_ImageMedianFilterMxN_Fp(3MLIB),
mllib_ImageMedianFilterMxN_US(3MLIB),mllib_ImageMinFilter3x3(3MLIB),
mllib_ImageMinFilter3x3_Fp(3MLIB),mllib_ImageMinFilter5x5(3MLIB),
mllib_ImageMinFilter5x5_Fp(3MLIB),mllib_ImageMinFilter7x7(3MLIB),
mllib_ImageMinFilter7x7_Fp(3MLIB),mllib_ImageRankFilter3x3(3MLIB),
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMedianFilter5x5\_US – 5x5 median filter, unsigned

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter5x5_US(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge, mllib_s32 bits);
```

**Description** The `mllib_ImageMedianFilter5x5_US()` function performs median filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
`MLIB_MEDIAN_MASK_RECT`  
`MLIB_MEDIAN_MASK_PLUS`  
`MLIB_MEDIAN_MASK_X`  
`MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
`MLIB_EDGE_DST_NO_WRITE`  
`MLIB_EDGE_DST_FILL_ZERO`  
`MLIB_EDGE_DST_COPY_SRC`  
`MLIB_EDGE_SRC_EXTEND`
- bits* The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`,  
`mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,  
`mllib_ImageRankFilter5x5_US(3MLIB)`, `mllib_ImageRankFilter7x7(3MLIB)`,  
`mllib_ImageRankFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter7x7_US(3MLIB)`,  
`mllib_ImageRankFilterMxN(3MLIB)`, `mllib_ImageRankFilterMxN_Fp(3MLIB)`,  
`mllib_ImageRankFilterMxN_US(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMedianFilter7x7 – 7x7 median filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMedianFilter7x7(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter7x7()` function performs median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),

---

```
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageMedianFilter7x7\_Fp – 7x7 median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilter7x7_Fp(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilter7x7_Fp()` function performs floating-point median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),

```
mllib_ImageMedianFilter3x3_US(3MLIB),mllib_ImageMedianFilter5x5(3MLIB),
mllib_ImageMedianFilter5x5_Fp(3MLIB),mllib_ImageMedianFilter5x5_US(3MLIB),
mllib_ImageMedianFilter7x7(3MLIB),mllib_ImageMedianFilter7x7_US(3MLIB),
mllib_ImageMedianFilterMxN(3MLIB),mllib_ImageMedianFilterMxN_Fp(3MLIB),
mllib_ImageMedianFilterMxN_US(3MLIB),mllib_ImageMinFilter3x3(3MLIB),
mllib_ImageMinFilter3x3_Fp(3MLIB),mllib_ImageMinFilter5x5(3MLIB),
mllib_ImageMinFilter5x5_Fp(3MLIB),mllib_ImageMinFilter7x7(3MLIB),
mllib_ImageMinFilter7x7_Fp(3MLIB),mllib_ImageRankFilter3x3(3MLIB),
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMedianFilter7x7\_US – 7x7 median filter, unsigned

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMedianFilter7x7_US(mllib_image *dst,
 const mllib_image *src, mllib_median_mask mmask, mllib_s32 cmask,
 mllib_edge edge, mllib_s32 bits);
```

**Description** The `mllib_ImageMedianFilter7x7_US()` function performs median filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
`MLIB_MEDIAN_MASK_RECT`  
`MLIB_MEDIAN_MASK_PLUS`  
`MLIB_MEDIAN_MASK_X`  
`MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
`MLIB_EDGE_DST_NO_WRITE`  
`MLIB_EDGE_DST_FILL_ZERO`  
`MLIB_EDGE_DST_COPY_SRC`  
`MLIB_EDGE_SRC_EXTEND`
- bits* The number of unsigned bits for pixel dynamic range.  $9 \leq bits \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,  
`mllib_ImageRankFilter5x5_US(3MLIB)`, `mllib_ImageRankFilter7x7(3MLIB)`,  
`mllib_ImageRankFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter7x7_US(3MLIB)`,  
`mllib_ImageRankFilterMxN(3MLIB)`, `mllib_ImageRankFilterMxN_Fp(3MLIB)`,  
`mllib_ImageRankFilterMxN_US(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMedianFilterMxN – MxN median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilterMxN(mllib_image *dst,
 const mllib_image *src, mllib_s32 m, mllib_s32 n, mllib_median_mask mmask,
 mllib_s32 cmask, mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilterMxN()` function performs MxN median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

*dst*            Pointer to destination image.

*src*            Pointer to source image.

*m*              Width of the filter window. *m* must be odd number greater than 1.

*n*              Height of the filter window. *n* must be odd number greater than 1.

*mmask*        Shape of the mask to be used for median filtering. It can be one of the following:

                 MLIB\_MEDIAN\_MASK\_RECT  
                 MLIB\_MEDIAN\_MASK\_PLUS  
                 MLIB\_MEDIAN\_MASK\_X  
                 MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE

*cmask*        Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.

*edge*         Type of edge condition. It can be one of the following:

                 MLIB\_EDGE\_DST\_NO\_WRITE  
                 MLIB\_EDGE\_DST\_FILL\_ZERO  
                 MLIB\_EDGE\_DST\_COPY\_SRC  
                 MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`,  
`mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,  
`mllib_ImageRankFilter5x5_US(3MLIB)`, `mllib_ImageRankFilter7x7(3MLIB)`,  
`mllib_ImageRankFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter7x7_US(3MLIB)`,  
`mllib_ImageRankFilterMxN(3MLIB)`, `mllib_ImageRankFilterMxN_Fp(3MLIB)`,  
`mllib_ImageRankFilterMxN_US(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMedianFilterMxN\_Fp – MxN median filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilterMxN_Fp(mllib_image *dst,
 const mllib_image *src, mllib_s32 m, mllib_s32 n, mllib_median_mask mmask,
 mllib_s32 cmask, mllib_edge edge);
```

**Description** The `mllib_ImageMedianFilterMxN_Fp()` function performs MxN median filtering on a floating-point image. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- m* Width of the filter window. *m* must be odd number greater than 1.
- n* Height of the filter window. *n* must be odd number greater than 1.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mlib_ImageMedianFilter3x3_US(3MLIB)`, `mlib_ImageMedianFilter5x5(3MLIB)`,  
`mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`,  
`mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mlib_ImageMedianFilter7x7_US(3MLIB)`, `mlib_ImageMedianFilterMxN(3MLIB)`,  
`mlib_ImageMedianFilterMxN_US(3MLIB)`, `mlib_ImageMinFilter3x3(3MLIB)`,  
`mlib_ImageMinFilter3x3_Fp(3MLIB)`, `mlib_ImageMinFilter5x5(3MLIB)`,  
`mlib_ImageMinFilter5x5_Fp(3MLIB)`, `mlib_ImageMinFilter7x7(3MLIB)`,  
`mlib_ImageMinFilter7x7_Fp(3MLIB)`, `mlib_ImageRankFilter3x3(3MLIB)`,  
`mlib_ImageRankFilter3x3_Fp(3MLIB)`, `mlib_ImageRankFilter3x3_US(3MLIB)`,  
`mlib_ImageRankFilter5x5(3MLIB)`, `mlib_ImageRankFilter5x5_Fp(3MLIB)`,  
`mlib_ImageRankFilter5x5_US(3MLIB)`, `mlib_ImageRankFilter7x7(3MLIB)`,  
`mlib_ImageRankFilter7x7_Fp(3MLIB)`, `mlib_ImageRankFilter7x7_US(3MLIB)`,  
`mlib_ImageRankFilterMxN(3MLIB)`, `mlib_ImageRankFilterMxN_Fp(3MLIB)`,  
`mlib_ImageRankFilterMxN_US(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMedianFilterMxN\_US – MxN median filter, unsigned

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMedianFilterMxN_US(mllib_image *dst,
 const mllib_image *src, mllib_s32 m, mllib_s32 n, mllib_median_mask mmask,
 mllib_s32 cmask, mllib_edge edge, mllib_s32 bits);
```

**Description** The `mllib_ImageMedianFilterMxN_US()` function performs MxN median filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- m* Width of the filter window. *m* must be odd number greater than 1.
- n* Height of the filter window. *n* must be odd number greater than 1.
- mmask* Shape of the mask to be used for median filtering. It can be one of the following:  
MLIB\_MEDIAN\_MASK\_RECT  
MLIB\_MEDIAN\_MASK\_PLUS  
MLIB\_MEDIAN\_MASK\_X  
MLIB\_MEDIAN\_MASK\_RECT\_SEPARABLE
- cmask* Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- edge* Type of edge condition. It can be one of the following:  
MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_DST\_COPY\_SRC  
MLIB\_EDGE\_SRC\_EXTEND
- bits* The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB), mllib_ImageMaxFilter3x3_Fp(3MLIB),  
mllib_ImageMaxFilter5x5(3MLIB), mllib_ImageMaxFilter5x5_Fp(3MLIB),  
mllib_ImageMaxFilter7x7(3MLIB), mllib_ImageMaxFilter7x7_Fp(3MLIB),  
mllib_ImageMedianFilter3x3(3MLIB), mllib_ImageMedianFilter3x3_Fp(3MLIB),  
mllib_ImageMedianFilter3x3_US(3MLIB), mllib_ImageMedianFilter5x5(3MLIB),  
mllib_ImageMedianFilter5x5_Fp(3MLIB), mllib_ImageMedianFilter5x5_US(3MLIB),  
mllib_ImageMedianFilter7x7(3MLIB), mllib_ImageMedianFilter7x7_Fp(3MLIB),  
mllib_ImageMedianFilter7x7_US(3MLIB), mllib_ImageMedianFilterMxN(3MLIB),  
mllib_ImageMedianFilterMxN_Fp(3MLIB), mllib_ImageMinFilter3x3(3MLIB),  
mllib_ImageMinFilter3x3_Fp(3MLIB), mllib_ImageMinFilter5x5(3MLIB),  
mllib_ImageMinFilter5x5_Fp(3MLIB), mllib_ImageMinFilter7x7(3MLIB),  
mllib_ImageMinFilter7x7_Fp(3MLIB), mllib_ImageRankFilter3x3(3MLIB),  
mllib_ImageRankFilter3x3_Fp(3MLIB), mllib_ImageRankFilter3x3_US(3MLIB),  
mllib_ImageRankFilter5x5(3MLIB), mllib_ImageRankFilter5x5_Fp(3MLIB),  
mllib_ImageRankFilter5x5_US(3MLIB), mllib_ImageRankFilter7x7(3MLIB),  
mllib_ImageRankFilter7x7_Fp(3MLIB), mllib_ImageRankFilter7x7_US(3MLIB),  
mllib_ImageRankFilterMxN(3MLIB), mllib_ImageRankFilterMxN_Fp(3MLIB),  
mllib_ImageRankFilterMxN_US(3MLIB), attributes(5)`

**Name** mllib\_ImageMin – two-image minimum

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMin(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMin()` function accepts input from the two source images and writes the minimum to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \text{MIN}\{ src1[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src1*     Pointer to first source image.
- src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMin\\_Fp\(3MLIB\)](#), [mllib\\_ImageMin\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageMin\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMinFilter3x3 – 3x3 Min Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMinFilter3x3(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMinFilter3x3()` function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where  $x = 1, \dots, w - 2$ ;  $y = 1, \dots, h - 2$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3_Fp(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```



**Name** mllib\_ImageMinFilter3x3\_Fp – 3x3 Min Filter, floating point

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMinFilter3x3_Fp(mllib_image *dst,
 const mllib_image *src);
```

**Description** The `mllib_ImageMinFilter3x3_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}$$

where  $x = 1, \dots, w - 2$ ;  $y = 1, \dots, h - 2$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter5x5(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMinFilter5x5 – 5x5 Min Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMinFilter5x5(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMinFilter5x5()` function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-2 \leq p \leq x+2; y-2 \leq q \leq y+2 \}$$

where  $x = 2, \dots, w - 3$ ;  $y = 2, \dots, h - 3$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5_Fp(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMinFilter5x5\_Fp – 5x5 Min Filter, floating point

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMinFilter5x5_Fp(mllib_image *dst,
 const mllib_image *src);
```

**Description** The `mllib_ImageMinFilter5x5_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-2 \leq p \leq x+2; y-2 \leq q \leq y+2 \}$$

where  $x = 2, \dots, w - 3$ ;  $y = 2, \dots, h - 3$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5(3MLIB)`, `mllib_ImageMinFilter7x7(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMinFilter7x7 – 7x7 Min Filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMinFilter7x7(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageMinFilter7x7()` function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-3 \leq p \leq x+3; y-3 \leq q \leq y+3 \}$$

where  $x = 3, \dots, w - 4$ ;  $y = 3, \dots, h - 4$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5(3MLIB)`, `mllib_ImageMinFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMinFilter7x7_Fp(3MLIB)`, `mllib_ImageRankFilter3x3(3MLIB)`,  
`mllib_ImageRankFilter3x3_Fp(3MLIB)`, `mllib_ImageRankFilter3x3_US(3MLIB)`,  
`mllib_ImageRankFilter5x5(3MLIB)`, `mllib_ImageRankFilter5x5_Fp(3MLIB)`,

```
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```



**Name** mlib\_ImageMinFilter7x7\_Fp – 7x7 Min Filter, floating point

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_ImageMinFilter7x7_Fp(mlib_image *dst,
 const mlib_image *src);
```

**Description** The `mlib_ImageMinFilter7x7_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \\ x-3 \leq p \leq x+3; y-3 \leq q \leq y+3 \}$$

where  $x = 3, \dots, w - 4$ ;  $y = 3, \dots, h - 4$ .

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mlib_ImageMedianFilter3x3_US(3MLIB)`, `mlib_ImageMedianFilter5x5(3MLIB)`,  
`mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`,  
`mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mlib_ImageMedianFilter7x7_US(3MLIB)`, `mlib_ImageMedianFilterMxN(3MLIB)`,  
`mlib_ImageMedianFilterMxN_Fp(3MLIB)`, `mlib_ImageMedianFilterMxN_US(3MLIB)`,  
`mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mlib_ImageMinFilter5x5(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`,  
`mlib_ImageMinFilter7x7(3MLIB)`, `mlib_ImageRankFilter3x3(3MLIB)`,

```
mllib_ImageRankFilter3x3_Fp(3MLIB),mllib_ImageRankFilter3x3_US(3MLIB),
mllib_ImageRankFilter5x5(3MLIB),mllib_ImageRankFilter5x5_Fp(3MLIB),
mllib_ImageRankFilter5x5_US(3MLIB),mllib_ImageRankFilter7x7(3MLIB),
mllib_ImageRankFilter7x7_Fp(3MLIB),mllib_ImageRankFilter7x7_US(3MLIB),
mllib_ImageRankFilterMxN(3MLIB),mllib_ImageRankFilterMxN_Fp(3MLIB),
mllib_ImageRankFilterMxN_US(3MLIB),attributes(5)
```

**Name** mllib\_ImageMin\_Fp – two-image minimum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMin_Fp(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMin_Fp()` function accepts input from the two floating-point source images and writes the minimum to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \text{MIN}\{ src1[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMin\(3MLIB\)](#), [mllib\\_ImageMin\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageMin\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMin\_Fp\_Inp – two-image minimum

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMin_Fp_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMin_Fp_Inp()` function accepts input from the two source images and writes the minimum in place on a pixel-by-pixel basis.

It uses the following equation:

`src1dst[x][y][i] = MIN{ src1dst[x][y][i], src2[x][y][i] }`

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to source and destination image.  
*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMin\(3MLIB\)](#), [mllib\\_ImageMin\\_Fp\(3MLIB\)](#), [mllib\\_ImageMin\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMinimum – image minimum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMinimum(mllib_s32 *min, const mllib_image *img);`

**Description** The `mllib_ImageMinimum()` function determines the minimum value for each channel in an image.

It uses the following equation:

$$\text{min}[i] = \text{MIN}\{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \}$$

**Parameters** The function takes the following arguments:

*min*      Pointer to minimum vector, where length is the number of channels in the image.  
          `min[i]` contains the minimum of channel `i`.

*img*      Pointer to a source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaximum\(3MLIB\)](#), [mllib\\_ImageMaximum\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinimum\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMinimum\_Fp – image minimum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageMinimum_Fp(mllib_d64 *min, const mllib_image *img);`

**Description** The `mllib_ImageMinimum_Fp()` function determines the minimum value for each channel in a floating-point image.

It uses the following equation:

$$\text{min}[i] = \text{MIN}\{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \}$$

**Parameters** The function takes the following arguments:

*min*     Pointer to minimum vector, where length is the number of channels in the image.  
         `min[i]` contains the minimum of channel `i`.

*img*     Pointer to a source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaximum\(3MLIB\)](#), [mllib\\_ImageMaximum\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinimum\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMin\_Inp – two-image minimum

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageMin_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMin_Inp()` function accepts input from the two source images and writes the minimum in place on a pixel-by-pixel basis.

It uses the following equation:

$$src1dst[x][y][i] = \text{MIN}\{ src1dst[x][y][i], src2[x][y][i] \}$$

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_ImageMin\(3MLIB\)](#), [mllib\\_ImageMin\\_Fp\(3MLIB\)](#), [mllib\\_ImageMin\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMoment2 – second moment

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMoment2(mllib_d64 *moment, const mllib_image *img);
```

**Description** The `mllib_ImageMoment2()` function computes the second moment of each channel in an image.

It uses the following equation:

$$\text{moment}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]**2$$

**Parameters** The function takes the following arguments:

- moment*      Pointer to moment array, where length is the number of channels in the image.
- img*            Pointer to an image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMoment2\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageMoment2\_Fp – second moment

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMoment2_Fp(mllib_d64 *moment, const mllib_image *img);`

**Description** The `mllib_ImageMoment2_Fp()` function computes the second moment of each channel in a floating-point image.

It uses the following equation:

$$\text{moment}[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]**2$$

**Parameters** The function takes the following arguments:

- moment*      Pointer to moment array, where length is the number of channels in the image.
- img*            Pointer to an image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMoment2\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMulAlpha – alpha channel multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMulAlpha(mllib_image *dst, const mllib_image *src,
 mllib_s32 cmask);
```

**Description** The `mllib_ImageMulAlpha()` function multiplies color channels by the alpha channel on a pixel by pixel basis.

For the `MLIB_BYTE` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] * \text{src}[x][y][a] * 2^{**(-8)}$$

For the `MLIB_SHORT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] * \text{src}[x][y][a] * 2^{**(-15)}$$

For the `MLIB_USHORT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] * \text{src}[x][y][a] * 2^{**(-16)}$$

For the `MLIB_INT` image, it uses the following equation:

$$\text{dst}[x][y][c] = \text{src}[x][y][c] * \text{src}[x][y][a] * 2^{**(-31)}$$

where `c` and `a` are the indices for the color channels and the alpha channel, respectively, so `c != a`.

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src*      Pointer to source image.
- cmask*    Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of *cmask* is the alpha channel.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageMulAlpha_Inp(3MLIB)`, `mlib_ImageMulAlpha_Fp(3MLIB)`,  
`mlib_ImageMulAlpha_Fp_Inp(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageMulAlpha\_Fp – alpha channel multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMulAlpha_Fp(mllib_image *dst, const mllib_image *src,
 mllib_s32 cmask);
```

**Description** The `mllib_ImageMulAlpha_Fp()` function multiplies floating-point color channels by the alpha channel on a pixel by pixel basis.

It uses the following equation:

$$dst[x][y][c] = src[x][y][c] * src[x][y][a]$$

where *c* and *a* are the indices for the color channels and the alpha channel, respectively, so *c* != *a*.

**Parameters** The function takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to source image.
- cmask*          Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of *cmask* is the alpha channel.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMulAlpha\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Inp\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMulAlpha\_Fp\_Inp – alpha channel multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMulAlpha_Fp_Inp(mllib_image *srcdst, mllib_s32 cmask);`

**Description** The `mllib_ImageMulAlpha_Fp_Inp()` function multiplies floating-point color channels by the alpha channel on a pixel by pixel basis, in place.

It uses the following equation:

$$srcdst[x][y][c] = srcdst[x][y][c] * srcdst[x][y][a]$$

where *c* and *a* are the indices for the color channels and the alpha channel, respectively, so *c* != *a*.

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- cmask*     Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of *cmask* is the alpha channel.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMulAlpha\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Inp\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMulAlpha\_Inp – alpha channel multiplication, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageMulAlpha_Inp(mllib_image *srcdst, mllib_s32 cmask);`

**Description** The `mllib_ImageMulAlpha_Inp()` function multiplies color channels by the alpha channel on a pixel by pixel basis, in place.

For the `MLIB_BYTE` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] * \text{srcdst}[x][y][a] * 2^{**(-8)}$$

For the `MLIB_SHORT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] * \text{srcdst}[x][y][a] * 2^{**(-15)}$$

For the `MLIB_USHORT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] * \text{srcdst}[x][y][a] * 2^{**(-16)}$$

For the `MLIB_INT` image, it uses the following equation:

$$\text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] * \text{srcdst}[x][y][a] * 2^{**(-31)}$$

where *c* and *a* are the indices for the color channels and the alpha channel, respectively, so *c* != *a*.

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- cmask*     Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of *cmask* is the alpha channel.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMulAlpha\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Fp\(3MLIB\)](#), [mllib\\_ImageMulAlpha\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMul\_Fp – computes the multiplication of two images on a pixel-by-pixel basis

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMul_Fp(mllib_image dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageMul_Fp()` function computes the multiplication of two floating-point images on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] * src2[x][y][i]$$

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMul\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMul\_Fp\_Inp – computes the multiplication of two images on a pixel-by-pixel basis

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageMul_Fp_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The mllib\_ImageMul\_Fp\_Inp() function computes the multiplication of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:

$src1dst[x][y][i] = src1dst[x][y][i] * src2[x][y][i]$

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMul\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_ImageMulShift – multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_ImageMulShift(mlib_image *dst, const mlib_image *src1,
 const mlib_image *src2, mlib_s32 shift);
```

**Description** The `mlib_ImageMulShift()` function multiplies the pixel values of the two source images. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] * src2[x][y][i] * 2^{*(-shift)}$$

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src1* Pointer to first source image.

*src2* Pointer to second source image.

*shift* Right shifting factor.  $0 \leq shift \leq 31$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageMulShift\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageMulShift\_Inp – multiplication, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageMulShift_Inp(mllib_image *src1dst, const mllib_image *src2,
 mllib_s32 shift);
```

**Description** The `mllib_ImageMulShift_Inp()` function multiplies the pixel values of the two source images in place. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] * \text{src2}[x][y][i] * 2^{*(-\text{shift})}$$

**Parameters** The function takes the following arguments:

- src1dst*     Pointer to source and destination image.
- src2*        Pointer to second source image.
- shift*        Right shifting factor.  $0 \leq \text{shift} \leq 31$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMulShift\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNormCrossCorrel – normalized cross correlation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageNormCrossCorrel(mllib_d64 *correl,
 const mllib_image *img1, const mllib_image *img2, const mllib_d64 *mean2,
 const mllib_d64 *sdev2);
```

**Description** The `mllib_ImageNormCrossCorrel()` function computes the normalized cross-correlation coefficients between a pair of images, on a per-channel basis.

It uses the following equations:

$$\text{correl}[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (d1[x][y][i] * d2[x][y][i])}{s1[i] * s2[i]}$$

$$d1[x][y][i] = \text{img1}[x][y][i] - m1[i]$$

$$d2[x][y][i] = \text{img2}[x][y][i] - m2[i]$$

$$m1[i] = \frac{1}{w*h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img1}[x][y][i]$$

$$m2[i] = \frac{1}{w*h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img2}[x][y][i]$$

$$s1[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] - m1[i])**2}$$

$$s2[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img2}[x][y][i] - m2[i])**2}$$

where `w` and `h` are the width and height of the images, respectively; `m1` and `m2` are the mean arrays of the first and second images, respectively; `s1` and `s2` are the un-normalized standard deviation arrays of the first and second images, respectively.

In usual cases, the normalized cross-correlation coefficient is in the range of `[-1.0, 1.0]`. In the case of `(s1[i] == 0)` or `(s2[i] == 0)`, where a constant image channel is involved, the normalized cross-correlation coefficient is defined as follows:

```
#define signof(x) ((x > 0) ? 1 : ((x < 0) ? -1 : 0))
```

```
if ((s1[i] == 0.) || (s2[i] == 0.)) {
 if ((s1[i] == 0.) && (s2[i] == 0.)) {
 if (signof(m1[i]) == signof(m2[i])) {
 correl[i] = 1.0;
 } else {
 correl[i] = -1.0;
 }
 } else {
 correl[i] = -1.0;
 }
}
```

The two images must have the same type, the same size, and the same number of channels. They can have 1, 2, 3 or 4 channels. They can be of type MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT or MLIB\_INT.

If (mean2 == NULL) or (sdev2 == NULL), then m2 and s2 are calculated in this function according to the formulas shown above. Otherwise, they are calculated as follows:

```
m2[i] = mean2[i];
s2[i] = sdev2[i] * sqrt(w*h);
```

where mean2 and sdev2 can be the output of mllib\_ImageMean() and mllib\_ImageStdDev(), respectively.

**Parameters** The function takes the following arguments:

- correl* Pointer to normalized cross correlation array on a channel basis. The array must be the size of channels in the images. *correl[i]* contains the cross-correlation of channel *i*.
- img1* Pointer to first image.
- img2* Pointer to second image.
- mean2* Pointer to the mean array of the second image.
- sdev2* Pointer to the standard deviation array of the second image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageAutoCorrel(3MLIB)`, `mlib_ImageAutoCorrel_Fp(3MLIB)`,  
`mlib_ImageCrossCorrel(3MLIB)`, `mlib_ImageCrossCorrel_Fp(3MLIB)`,  
`mlib_ImageNormCrossCorrel_Fp(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageNormCrossCorrel\_Fp – normalized cross correlation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageNormCrossCorrel_Fp(mllib_d64 *correl,
 const mllib_image *img1, const mllib_image *img2, const mllib_d64 *mean2,
 const mllib_d64 *sdev2);
```

**Description** The `mllib_ImageNormCrossCorrel_Fp()` function computes the normalized cross-correlation coefficients between a pair of floating-point images, on a per-channel basis.

It uses the following equations:

$$\text{correl}[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (d1[x][y][i] * d2[x][y][i])}{s1[i] * s2[i]}$$

$$d1[x][y][i] = \text{img1}[x][y][i] - m1[i]$$

$$d2[x][y][i] = \text{img2}[x][y][i] - m2[i]$$

$$m1[i] = \frac{1}{w*h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img1}[x][y][i]$$

$$m2[i] = \frac{1}{w*h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img2}[x][y][i]$$

$$s1[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] - m1[i])**2}$$

$$s2[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img2}[x][y][i] - m2[i])**2}$$

where *w* and *h* are the width and height of the images, respectively; *m1* and *m2* are the mean arrays of the first and second images, respectively; *s1* and *s2* are the un-normalized standard deviation arrays of the first and second images, respectively.

In usual cases, the normalized cross-correlation coefficient is in the range of `[-1.0, 1.0]`. In the case of `(s1[i] == 0)` or `(s2[i] == 0)`, where a constant image channel is involved, the normalized cross-correlation coefficient is defined as follows:

```
#define signof(x) ((x > 0) ? 1 : ((x < 0) ? -1 : 0))
```

```

if ((s1[i] == 0.) || (s2[i] == 0.)) {
 if ((s1[i] == 0.) && (s2[i] == 0.)) {
 if (signof(m1[i]) == signof(m2[i])) {
 correl[i] = 1.0;
 } else {
 correl[i] = -1.0;
 }
 } else {
 correl[i] = -1.0;
 }
}
}

```

The two images must have the same type, the same size, and the same number of channels. They can have 1, 2, 3 or 4 channels. They can be of type `MLIB_FLOAT` or `MLIB_DOUBLE`.

If (`mean2 == NULL`) or (`sdev2 == NULL`), then `m2` and `s2` are calculated in this function according to the formulas shown above. Otherwise, they are calculated as follows:

```

m2[i] = mean2[i];
s2[i] = sdev2[i] * sqrt(w*h);

```

where `mean2` and `sdev2` can be the output of `mllib_ImageMean()` and `mllib_ImageStdDev()`, respectively.

In some cases, the resulting coefficients of this function could be NaN, Inf, or -Inf.

**Parameters** The function takes the following arguments:

*correl*     Pointer to normalized cross correlation array on a channel basis. The array must be the size of channels in the images. `correl[i]` contains the cross-correlation of channel `i`.

*img1*       Pointer to first image.

*img2*       Pointer to second image.

*mean2*      Pointer to the mean array of the second image.

*sdev2*      Pointer to the standard deviation array of the second image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageAutoCorrel(3MLIB)`, `mllib_ImageAutoCorrel_Fp(3MLIB)`,  
`mllib_ImageCrossCorrel(3MLIB)`, `mllib_ImageCrossCorrel_Fp(3MLIB)`,  
`mllib_ImageNormCrossCorrel(3MLIB)`, `attributes(5)`



**Name** mllib\_ImageNot – Not

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageNot(mllib_image *dst, const mllib_image *src);`

**Description** The `mllib_ImageNot()` function computes the logical Not of each pixel in the source image.

It uses the following equation:

$$dst[x][y][i] = \sim src[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNot\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNotAnd – NotAnd

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageNotAnd(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageNotAnd()` function computes the logical And of the first source image with the second source image and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \sim(src1[x][y][i] \& (src2[x][y][i]))$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src1*     Pointer to first source image.
- src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNotAnd\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageNotAnd\_Inp – NotAnd, in place

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_ImageNotAnd_Inp(mlib_image *src1dst,
 const mlib_image *src2);
```

**Description** The `mlib_ImageNotAnd_Inp()` function computes the logical And of the first source image with the second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

$$src1dst[x][y][i] = \sim(src1dst[x][y][i] \& (src2[x][y][i]))$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageNotAnd\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNot\_Inp – Not

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageNot_Inp(mllib_image *srcdst);`

**Description** The `mllib_ImageNot_Inp()` function computes the logical Not of each pixel in the source image, in place.

It uses the following equation:

`srcdst[x][y][i] = ~srcdst[x][y][i]`

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

`srcdst`     Pointer to source and destination image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNot\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNotOr – NotOr

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageNotOr(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageNotOr()` function computes the logical Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = \sim(src1[x][y][i] \mid src2[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNotOr\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNotOr\_Inp – NotOr, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageNotOr_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageNotOr_Inp()` function computes the logical Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \sim(\text{src1dst}[x][y][i] \mid \text{src2}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNotOr\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNotXor – NotXor

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageNotXor(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageNotXor()` function computes the logical exclusive Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = \sim(\text{src1}[x][y][i] \wedge \text{src2}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNotXor\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageNotXor\_Inp – NotXor, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageNotXor_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageNotXor_Inp()` function computes the logical exclusive Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \sim(\text{src1dst}[x][y][i] \wedge \text{src2}[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageNotXor\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageOr – Or

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageOr(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageOr()` function computes the logical Or of the first source image with the second source image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] \mid src2[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageOr\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageOr\_Inp – Or, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageOr_Inp(mllib_image *src1dst, const mllib_image *src2);`

**Description** The `mllib_ImageOr_Inp()` function computes the logical Or of the first source image with the second source image on a pixel-by-pixel basis, in place.

It uses the following equation:

`src1dst[x][y][i] = src1dst[x][y][i] | src2[x][y][i]`

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageOr\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageOrNot1\_Inp – OrNot, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageOrNot1_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageOrNot1_Inp()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$src1dst[x][y][i] = src1dst[x][y][i] \mid (\sim src2[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageOrNot\(3MLIB\)](#), [mllib\\_ImageOrNot2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageOrNot2\_Inp – OrNot, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageOrNot2_Inp(mllib_image *src2dst,
 const mllib_image *src1);
```

**Description** The `mllib_ImageOrNot2_Inp()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$src2dst[x][y][i] = src1[x][y][i] \mid (\sim src2dst[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src2dst*      Pointer to second source and destination image.

*src1*          Pointer to first source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageOrNot\(3MLIB\)](#), [mllib\\_ImageOrNot1\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageOrNot – OrNot

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageOrNot(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageOrNot()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] \mid (\sim src2[x][y][i])$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src1*     Pointer to first source image.
- src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageOrNot1\\_Inp\(3MLIB\)](#), [mllib\\_ImageOrNot2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImagePolynomialWarp – polynomial-based image warp

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImagePolynomialWarp(mllib_image *dst,
 const mllib_image *src, const mllib_d64 *xCoeffs,
 const mllib_d64 *yCoeffs, mllib_s32 n, mllib_d64 preShiftX,
 mllib_d64 preShiftY, mllib_d64 postShiftX, mllib_d64 postShiftY,
 mllib_d64 preScaleX, mllib_d64 preScaleY, mllib_d64 postScaleX,
 mllib_d64 postScaleY, mllib_filter filter, mllib_edge edge);
```

**Description** The `mllib_ImagePolynomialWarp()` function performs a polynomial-based image warp.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The source and destination images may have different sizes.

The *xCoeffs* and *yCoeffs* parameters must contain the same number of coefficients of the form  $(n + 1)(n + 2)/2$  for some  $n$ , where  $n$  is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

```
1, x, y, x**2, x*y, y**2, ...,
x**n, x**(n-1)*y, ..., x*y**(n-1), y**n
```

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is backward mapped to a point S in the source image. Then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the *filter* parameter to generate the pixel value for point D.

The mapping is defined by the two bivariate polynomial functions  $X(x, y)$  and  $Y(x, y)$  that map destination (x, y) coordinates to source X and Y positions respectively.

The functions  $X(x, y)$  and  $Y(x, y)$  are:

$$\text{preX} = (x + \text{preShiftX}) * \text{preScaleX}$$
$$\text{preY} = (y + \text{preShiftY}) * \text{preScaleY}$$
$$\text{warpedX} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{xCoeffs}_{ij} * \text{preX}^{i-j} * \text{preY}^{j} \} \right\}$$
$$\text{warpedY} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{yCoeffs}_{ij} * \text{preX}^{i-j} * \text{preY}^{j} \} \right\}$$

i=0 j=0

$X(x, y) = \text{warpedX} * \text{postScaleX} - \text{postShiftX}$

$Y(x, y) = \text{warpedY} * \text{postScaleY} - \text{postShiftY}$

The destination (x, y) coordinates are pre-shifted by (preShiftX, preShiftY) and pre-scaled by the factors preScaleX and preScaleY prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by postScaleX and postScaleY, and then shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters** The function takes the following arguments:

|                   |                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>        | Pointer to destination image.                                                                                                |
| <i>src</i>        | Pointer to source image.                                                                                                     |
| <i>xCoeffs</i>    | Destination to source transform coefficients for the X coordinate.                                                           |
| <i>yCoeffs</i>    | Destination to source transform coefficients for the Y coordinate.                                                           |
| <i>n</i>          | Degree power of the polynomial.                                                                                              |
| <i>preShiftX</i>  | Displacement to apply to destination X positions.                                                                            |
| <i>preShiftY</i>  | Displacement to apply to destination Y positions.                                                                            |
| <i>postShiftX</i> | Displacement to apply to source X positions.                                                                                 |
| <i>postShiftY</i> | Displacement to apply to source Y positions.                                                                                 |
| <i>preScaleX</i>  | Scale factor to apply to destination X positions.                                                                            |
| <i>preScaleY</i>  | Scale factor to apply to destination Y positions.                                                                            |
| <i>postScaleX</i> | Scale factor to apply to source X positions.                                                                                 |
| <i>postScaleY</i> | Scale factor to apply to source Y positions.                                                                                 |
| <i>filter</i>     | Type of resampling filter. It can be one of the following:<br>MLIB_NEAREST<br>MLIB_BILINEAR<br>MLIB_BICUBIC<br>MLIB_BICUBIC2 |
| <i>edge</i>       | Type of edge condition. It can be one of the following:<br>MLIB_EDGE_DST_NO_WRITE<br>MLIB_EDGE_SRC_PADDED                    |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImagePolynomialWarp\\_Fp\(3MLIB\)](#), [mllib\\_ImagePolynomialWarpTable\(3MLIB\)](#), [mllib\\_ImagePolynomialWarpTable\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImagePolynomialWarp\_Fp – polynomial-based image warp

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImagePolynomialWarp_Fp(mllib_image *dst,
 const mllib_image *src, const mllib_d64 *xCoeffs,
 const mllib_d64 *yCoeffs, mllib_s32 n, mllib_d64 preShiftX,
 mllib_d64 preShiftY, mllib_d64 postShiftX, mllib_d64 postShiftY,
 mllib_d64 preScaleX, mllib_d64 preScaleY, mllib_d64 postScaleX,
 mllib_d64 postScaleY, mllib_filter filter, mllib_edge edge);
```

**Description** The `mllib_ImagePolynomialWarp_Fp()` function performs a polynomial-based image warp on a floating-point image.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The source and destination images may have different sizes.

The *xCoeffs* and *yCoeffs* parameters must contain the same number of coefficients of the form  $(n+1)(n+2)/2$  for some  $n$ , where  $n$  is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

1,  $x$ ,  $y$ ,  $x^2$ ,  $xy$ ,  $y^2$ , ...,  
 $x^n$ ,  $x^{n-1}y$ , ...,  $xy^{n-1}$ ,  $y^n$

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is backward mapped to a point S in the source image. Then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the *filter* parameter to generate the pixel value for point D.

The mapping is defined by the two bivariate polynomial functions  $X(x, y)$  and  $Y(x, y)$  that map destination  $(x, y)$  coordinates to source  $X$  and  $Y$  positions respectively.

The functions  $X(x, y)$  and  $Y(x, y)$  are:

$preX = (x + preShiftX) * preScaleX$

$preY = (y + preShiftY) * preScaleY$

$$warpedX = \sum_{i=0}^n \sum_{j=0}^i \{xCoeffs_{ij} * preX^{i-j} * preY^j\}$$

$$\text{warpedY} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{yCoeffs\_ij} * \text{preX}^{i-j} * \text{preY}^j \} \right\}$$
$$X(x, y) = \text{warpedX} * \text{postScaleX} - \text{postShiftX}$$
$$Y(x, y) = \text{warpedY} * \text{postScaleY} - \text{postShiftY}$$

The destination (x, y) coordinates are pre-shifted by (preShiftX, preShiftY) and pre-scaled by the factors preScaleX and preScaleY prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by postScaleX and postScaleY, and then shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters** The function takes the following arguments:

|                   |                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>        | Pointer to destination image.                                                                                                |
| <i>src</i>        | Pointer to source image.                                                                                                     |
| <i>xCoeffs</i>    | Destination to source transform coefficients for the X coordinate.                                                           |
| <i>yCoeffs</i>    | Destination to source transform coefficients for the Y coordinate.                                                           |
| <i>n</i>          | Degree power of the polynomial.                                                                                              |
| <i>preShiftX</i>  | Displacement to apply to destination X positions.                                                                            |
| <i>preShiftY</i>  | Displacement to apply to destination Y positions.                                                                            |
| <i>postShiftX</i> | Displacement to apply to source X positions.                                                                                 |
| <i>postShiftY</i> | Displacement to apply to source Y positions.                                                                                 |
| <i>preScaleX</i>  | Scale factor to apply to destination X positions.                                                                            |
| <i>preScaleY</i>  | Scale factor to apply to destination Y positions.                                                                            |
| <i>postScaleX</i> | Scale factor to apply to source X positions.                                                                                 |
| <i>postScaleY</i> | Scale factor to apply to source Y positions.                                                                                 |
| <i>filter</i>     | Type of resampling filter. It can be one of the following:<br>MLIB_NEAREST<br>MLIB_BILINEAR<br>MLIB_BICUBIC<br>MLIB_BICUBIC2 |
| <i>edge</i>       | Type of edge condition. It can be one of the following:<br>MLIB_EDGE_DST_NO_WRITE<br>MLIB_EDGE_SRC_PADDED                    |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImagePolynomialWarp\(3MLIB\)](#), [mllib\\_ImagePolynomialWarpTable\(3MLIB\)](#), [mllib\\_ImagePolynomialWarpTable\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImagePolynomialWarpTable – polynomial-based image warp with table-driven interpolation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImagePolynomialWarpTable(mllib_image *dst,
 const mllib_image *src, const mllib_d64 *xCoeffs,
 const mllib_d64 *yCoeffs, mllib_s32 n, mllib_d64 preShiftX,
 mllib_d64 preShiftY, mllib_d64 postShiftX, mllib_d64 postShiftY,
 mllib_d64 preScaleX, mllib_d64 preScaleY, mllib_d64 postScaleX,
 mllib_d64 postScaleY, const void *interp_table, mllib_edge edge);
```

**Description** The `mllib_ImagePolynomialWarpTable()` function performs a polynomial-based image warp with table-driven interpolation.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The source and destination images may have different sizes.

The *xCoeffs* and *yCoeffs* parameters must contain the same number of coefficients of the form  $(n+1)(n+2)/2$  for some  $n$ , where  $n$  is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

```
1, x, y, x**2, x*y, y**2, ...,
x**n, x**(n-1)*y, ..., x*y**(n-1), y**n
```

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is backward mapped to a point S in the source image. Then the source pixels with their centers surrounding point S are selected to do the interpolation specified by *interp\_table* to generate the pixel value for point D.

The mapping is defined by the two bivariate polynomial functions  $X(x, y)$  and  $Y(x, y)$  that map destination  $(x, y)$  coordinates to source  $X$  and  $Y$  positions respectively.

The functions  $X(x, y)$  and  $Y(x, y)$  are:

```
preX = (x + preShiftX)*preScaleX
```

```
preY = (y + preShiftY)*preScaleY
```

```

 n i
warpedX = SUM {SUM {xCoeffs_ij * preX**(i-j) * preY**j}}
 i=0 j=0
```

$$\text{warpedY} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{yCoeffs\_ij} * \text{preX}^{i-j} * \text{preY}^j \} \right\}$$

$$X(x, y) = \text{warpedX} * \text{postScaleX} - \text{postShiftX}$$

$$Y(x, y) = \text{warpedY} * \text{postScaleY} - \text{postShiftY}$$

The destination (x, y) coordinates are pre-shifted by (preShiftX, preShiftY) and pre-scaled by the factors preScaleX and preScaleY prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by postScaleX and postScaleY, and then shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters** The function takes the following arguments:

|                     |                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>          | Pointer to destination image.                                                                                        |
| <i>src</i>          | Pointer to source image.                                                                                             |
| <i>xCoeffs</i>      | Destination to source transform coefficients for the X coordinate.                                                   |
| <i>yCoeffs</i>      | Destination to source transform coefficients for the Y coordinate.                                                   |
| <i>n</i>            | Degree power of the polynomial.                                                                                      |
| <i>preShiftX</i>    | Displacement to apply to destination X positions.                                                                    |
| <i>preShiftY</i>    | Displacement to apply to destination Y positions.                                                                    |
| <i>postShiftX</i>   | Displacement to apply to source X positions.                                                                         |
| <i>postShiftY</i>   | Displacement to apply to source Y positions.                                                                         |
| <i>preScaleX</i>    | Scale factor to apply to destination X positions.                                                                    |
| <i>preScaleY</i>    | Scale factor to apply to destination Y positions.                                                                    |
| <i>postScaleX</i>   | Scale factor to apply to source X positions.                                                                         |
| <i>postScaleY</i>   | Scale factor to apply to source Y positions.                                                                         |
| <i>interp_table</i> | Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function. |
| <i>edge</i>         | Type of edge condition. It can be one of the following:<br>MLIB_EDGE_DST_NO_WRITE<br>MLIB_EDGE_SRC_PADDED            |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageInterpTableCreate\(3MLIB\)](#), [mllib\\_ImageInterpTableDelete\(3MLIB\)](#),  
[mllib\\_ImagePolynomialWarpTable\\_Fp\(3MLIB\)](#), [mllib\\_ImagePolynomialWarp\(3MLIB\)](#),  
[mllib\\_ImagePolynomialWarp\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImagePolynomialWarpTable\_Fp – polynomial-based image warp with table-driven interpolation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImagePolynomialWarpTable_Fp(mllib_image *dst,
 const mllib_image *src, const mllib_d64 *xCoeffs,
 const mllib_d64 *yCoeffs, mllib_s32 n, mllib_d64 preShiftX,
 mllib_d64 preShiftY, mllib_d64 postShiftX, mllib_d64 postShiftY,
 mllib_d64 preScaleX, mllib_d64 preScaleY, mllib_d64 postScaleX,
 mllib_d64 postScaleY, const void *interp_table, mllib_edge edge);
```

**Description** The `mllib_ImagePolynomialWarpTable_Fp()` function performs a polynomial-based image warp on a floating-point image with table-driven interpolation.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The source and destination images may have different sizes.

The *xCoeffs* and *yCoeffs* parameters must contain the same number of coefficients of the form  $(n+1)(n+2)/2$  for some  $n$ , where  $n$  is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

1,  $x$ ,  $y$ ,  $x^2$ ,  $xy$ ,  $y^2$ , ...,  
 $x^n$ ,  $x^{n-1}y$ , ...,  $xy^{n-1}$ ,  $y^n$

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point  $D$  is backward mapped to a point  $S$  in the source image. Then the source pixels with their centers surrounding point  $S$  are selected to do the interpolation specified by *interp\_table* to generate the pixel value for point  $D$ .

The mapping is defined by the two bivariate polynomial functions  $X(x, y)$  and  $Y(x, y)$  that map destination  $(x, y)$  coordinates to source  $X$  and  $Y$  positions respectively.

The functions  $X(x, y)$  and  $Y(x, y)$  are:

$$\text{preX} = (x + \text{preShiftX}) * \text{preScaleX}$$

$$\text{preY} = (y + \text{preShiftY}) * \text{preScaleY}$$

$$\text{warpedX} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{xCoeffs}_{ij} * \text{preX}^{i-j} * \text{preY}^{jj} \} \right\}$$

$$\text{warpedY} = \sum_{i=0}^n \left\{ \sum_{j=0}^i \{ \text{yCoeffs\_ij} * \text{preX}^{*(i-j)} * \text{preY}^{*j} \} \right\}$$

$$X(x, y) = \text{warpedX} * \text{postScaleX} - \text{postShiftX}$$

$$Y(x, y) = \text{warpedY} * \text{postScaleY} - \text{postShiftY}$$

The destination (x, y) coordinates are pre-shifted by (preShiftX, preShiftY) and pre-scaled by the factors preScaleX and preScaleY prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by postScaleX and postScaleY, and then shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters** The function takes the following arguments:

|                     |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>          | Pointer to destination image.                                                                                                           |
| <i>src</i>          | Pointer to source image.                                                                                                                |
| <i>xCoeffs</i>      | Destination to source transform coefficients for the X coordinate.                                                                      |
| <i>yCoeffs</i>      | Destination to source transform coefficients for the Y coordinate.                                                                      |
| <i>n</i>            | Degree power of the polynomial.                                                                                                         |
| <i>preShiftX</i>    | Displacement to apply to destination X positions.                                                                                       |
| <i>preShiftY</i>    | Displacement to apply to destination Y positions.                                                                                       |
| <i>postShiftX</i>   | Displacement to apply to source X positions.                                                                                            |
| <i>postShiftY</i>   | Displacement to apply to source Y positions.                                                                                            |
| <i>preScaleX</i>    | Scale factor to apply to destination X positions.                                                                                       |
| <i>preScaleY</i>    | Scale factor to apply to destination Y positions.                                                                                       |
| <i>postScaleX</i>   | Scale factor to apply to source X positions.                                                                                            |
| <i>postScaleY</i>   | Scale factor to apply to source Y positions.                                                                                            |
| <i>interp_table</i> | Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.                    |
| <i>edge</i>         | Type of edge condition. It can be one of the following:<br><br><code>MLIB_EDGE_DST_NO_WRITE</code><br><code>MLIB_EDGE_SRC_PADDED</code> |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.



**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageInterpTableCreate\(3MLIB\)](#), [mllib\\_ImageInterpTableDelete\(3MLIB\)](#),  
[mllib\\_ImagePolynomialWarpTable\(3MLIB\)](#), [mllib\\_ImagePolynomialWarp\(3MLIB\)](#),  
[mllib\\_ImagePolynomialWarp\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRankFilter3x3 – 3x3 rank filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter3x3(mllib_image *dst, const mllib_image *src,
 mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter3x3()` function performs 3x3 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),  
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),  
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#),  
[mllib\\_ImageRankFilter5x5\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\(3MLIB\)](#),  
[mllib\\_ImageRankFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\\_US\(3MLIB\)](#),

```
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter3x3\_Fp – 3x3 rank filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageRankFilter3x3_Fp(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter3x3_Fp()` function performs floating-point 3x3 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- rank* The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\(3MLIB\)](#),

```
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter3x3\_US – 3x3 rank filter, unsigned

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter3x3_US(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank, mllib_s32 bits);
```

**Description** The `mllib_ImageRankFilter3x3_US()` function performs 3x3 rank filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- bits*       The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),  
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),  
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter5x5\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#),

```
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter5x5 – 5x5 rank filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter5x5(mllib_image *dst, const mllib_image *src,
 mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter5x5()` function performs 5x5 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),  
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),  
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\(3MLIB\)](#),  
[mllib\\_ImageRankFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\\_US\(3MLIB\)](#),



```
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter5x5\_Fp – 5x5 rank filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageRankFilter5x5_Fp(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter5x5_Fp()` function performs floating-point 5x5 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- rank* The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\(3MLIB\)](#),

```
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter5x5\_US – 5x5 rank filter, unsigned

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter5x5_US(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank, mllib_s32 bits);
```

**Description** The `mllib_ImageRankFilter5x5_US()` function performs 5x5 rank filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- bits*       The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5(3MLIB)`, `mllib_ImageMinFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMinFilter7x7(3MLIB)`, `mllib_ImageMinFilter7x7_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3(3MLIB)`, `mllib_ImageRankFilter3x3_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3_US(3MLIB)`, `mllib_ImageRankFilter5x5(3MLIB)`,

```
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter7x7 – 7x7 rank filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter7x7(mllib_image *dst, const mllib_image *src,
 mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter7x7()` function performs 7x7 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),  
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),  
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageRankFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter7x7\\_US\(3MLIB\)](#),

```
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter7x7\_Fp – 7x7 rank filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageRankFilter7x7_Fp(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilter7x7_Fp()` function performs floating-point 7x7 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageRankFilter5x5\\_US\(3MLIB\)](#),



```
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilter7x7\_US – 7x7 rank filter, unsigned

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilter7x7_US(mllib_image *dst,
 const mllib_image *src, mllib_s32 rank, mllib_s32 bits);
```

**Description** The `mllib_ImageRankFilter7x7_US()` function performs 7x7 rank filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- rank*       The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- bits*       The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5(3MLIB)`, `mllib_ImageMinFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMinFilter7x7(3MLIB)`, `mllib_ImageMinFilter7x7_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3(3MLIB)`, `mllib_ImageRankFilter3x3_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3_US(3MLIB)`, `mllib_ImageRankFilter5x5(3MLIB)`,

```
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilterMxN – MxN rank filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilterMxN(mllib_image *dst, const mllib_image *src,
 mllib_s32 m, mllib_s32 n, mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilterMxN()` function performs MxN rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- m* Width of the filter window. *m* must be odd number greater than 1.
- n* Height of the filter window. *n* must be odd number greater than 1.
- rank* The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`,  
`mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`,  
`mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`,  
`mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`,  
`mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`,  
`mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,  
`mllib_ImageMinFilter5x5(3MLIB)`, `mllib_ImageMinFilter5x5_Fp(3MLIB)`,  
`mllib_ImageMinFilter7x7(3MLIB)`, `mllib_ImageMinFilter7x7_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3(3MLIB)`, `mllib_ImageRankFilter3x3_Fp(3MLIB)`,  
`mllib_ImageRankFilter3x3_US(3MLIB)`, `mllib_ImageRankFilter5x5(3MLIB)`,

```
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilterMxN\_Fp – MxN rank filter

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRankFilterMxN_Fp(mllib_image *dst,
 const mllib_image *src, mllib_s32 m, mllib_s32 n, mllib_s32 rank);
```

**Description** The `mllib_ImageRankFilterMxN_Fp()` function performs floating-point MxN rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- m* Width of the filter window. *m* must be odd number greater than 1.
- n* Height of the filter window. *n* must be odd number greater than 1.
- rank* The rank of the destination pixel. The pixel with minimum value is designated rank 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMaxFilter3x3\(3MLIB\)](#), [mllib\\_ImageMaxFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter5x5\(3MLIB\)](#), [mllib\\_ImageMaxFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMaxFilter7x7\(3MLIB\)](#), [mllib\\_ImageMaxFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\(3MLIB\)](#), [mllib\\_ImageMedianFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter3x3\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter5x5\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilter5x5\\_US\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\(3MLIB\)](#), [mllib\\_ImageMedianFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMedianFilter7x7\\_US\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\(3MLIB\)](#),  
[mllib\\_ImageMedianFilterMxN\\_Fp\(3MLIB\)](#), [mllib\\_ImageMedianFilterMxN\\_US\(3MLIB\)](#),  
[mllib\\_ImageMinFilter3x3\(3MLIB\)](#), [mllib\\_ImageMinFilter3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter5x5\(3MLIB\)](#), [mllib\\_ImageMinFilter5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageMinFilter7x7\(3MLIB\)](#), [mllib\\_ImageMinFilter7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRankFilter3x3\(3MLIB\)](#), [mllib\\_ImageRankFilter3x3\\_Fp\(3MLIB\)](#),

```
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
```

**Name** mllib\_ImageRankFilterMxN\_US – MxN rank filter, unsigned

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageRankFilterMxN_US(mllib_image *dst,
 const mllib_image *src, mllib_s32 m, mllib_s32 n, mllib_s32 rank,
 mllib_s32 bits);
```

**Description** The `mllib_ImageRankFilterMxN_US()` function performs MxN rank filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- m* Width of the filter window. *m* must be odd number greater than 1.
- n* Height of the filter window. *n* must be odd number greater than 1.
- rank* The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- bits* The number of unsigned bits for pixel dynamic range.  $9 \leq \text{bits} \leq 15$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageMaxFilter3x3(3MLIB)`, `mllib_ImageMaxFilter3x3_Fp(3MLIB)`, `mllib_ImageMaxFilter5x5(3MLIB)`, `mllib_ImageMaxFilter5x5_Fp(3MLIB)`, `mllib_ImageMaxFilter7x7(3MLIB)`, `mllib_ImageMaxFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3(3MLIB)`, `mllib_ImageMedianFilter3x3_Fp(3MLIB)`, `mllib_ImageMedianFilter3x3_US(3MLIB)`, `mllib_ImageMedianFilter5x5(3MLIB)`, `mllib_ImageMedianFilter5x5_Fp(3MLIB)`, `mllib_ImageMedianFilter5x5_US(3MLIB)`, `mllib_ImageMedianFilter7x7(3MLIB)`, `mllib_ImageMedianFilter7x7_Fp(3MLIB)`, `mllib_ImageMedianFilter7x7_US(3MLIB)`, `mllib_ImageMedianFilterMxN(3MLIB)`, `mllib_ImageMedianFilterMxN_Fp(3MLIB)`, `mllib_ImageMedianFilterMxN_US(3MLIB)`, `mllib_ImageMinFilter3x3(3MLIB)`, `mllib_ImageMinFilter3x3_Fp(3MLIB)`,



```
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB),
mlib_ImageRankFilterMxN_Fp(3MLIB), attributes(5)
```

**Name** mllib\_ImageReformat – image data buffer reformat

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageReformat(void **dstData, const void **srcData,
 mllib_s32 numBands, mllib_s32 xSize, mllib_s32 ySize,
 mllib_type dstDataType, const mllib_s32 *dstBandoffsets,
 mllib_s32 dstScanlinestride, mllib_s32 dstPixelstride,
 mllib_type srcDataType, const mllib_s32 *srcBandoffsets,
 mllib_s32 srcScanlinestride, mllib_s32 srcPixelstride);
```

**Description** The `mllib_ImageReformat()` function copies and casts, if needed, an image from one buffer to another. The formats and data types of the two buffers may be different.

```
dstPixel[x][y][i] = (dstDataType) srcPixel[x][y][i]
```

where the values of a pixel at position (x, y) and in channel i are:

```
srcPixel[x][y][i] = srcData[i][srcBandoffsets[i] +
 srcScanlinestride*y +
 srcPixelstride*x]
```

```
dstPixel[x][y][i] = dstData[i][dstBandoffsets[i] +
 dstScanlinestride*y +
 dstPixelstride*x]
```

It is the user's responsibility to make sure that the data buffers supplied are suitable for this operation. The `srcData` and `dstData` can have 1, 2, 3, or 4 channels, and they must have the same number of channels. The `srcDataType` and `dstDataType` can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The conversions between different data types are implemented as described in the following table:

| Source Type | Dest. Type | Action                             |
|-------------|------------|------------------------------------|
| MLIB_SHORT  | MLIB_BYTE  | (mllib_u8)clamp(x, 0, 255)         |
| MLIB_USHORT | MLIB_BYTE  | (mllib_u8)clamp(x, 0, 255)         |
| MLIB_INT    | MLIB_BYTE  | (mllib_u8)clamp(x, 0, 255)         |
| MLIB_FLOAT  | MLIB_BYTE  | (mllib_u8)clamp(x, 0, 255)         |
| MLIB_DOUBLE | MLIB_BYTE  | (mllib_u8)clamp(x, 0, 255)         |
| MLIB_BYTE   | MLIB_SHORT | (mllib_s16)x                       |
| MLIB_USHORT | MLIB_SHORT | (mllib_s16)clamp(x, -32768, 32767) |

| Source Type | Dest. Type  | Action                                         |
|-------------|-------------|------------------------------------------------|
| MLIB_INT    | MLIB_SHORT  | (mllib_s16)clamp(x, -32768, 32767)             |
| MLIB_FLOAT  | MLIB_SHORT  | (mllib_s16)clamp(x, -32768, 32767)             |
| MLIB_DOUBLE | MLIB_SHORT  | (mllib_s16)clamp(x, -32768, 32767)             |
| MLIB_BYTE   | MLIB_USHORT | (mllib_u16)x                                   |
| MLIB_SHORT  | MLIB_USHORT | (mllib_u16)clamp(x, 0, 65535)                  |
| MLIB_INT    | MLIB_USHORT | (mllib_u16)clamp(x, 0, 65535)                  |
| MLIB_FLOAT  | MLIB_USHORT | (mllib_u16)clamp(x, 0, 65535)                  |
| MLIB_DOUBLE | MLIB_USHORT | (mllib_u16)clamp(x, 0, 65535)                  |
| MLIB_BYTE   | MLIB_INT    | (mllib_s32)x                                   |
| MLIB_SHORT  | MLIB_INT    | (mllib_s32)x                                   |
| MLIB_USHORT | MLIB_INT    | (mllib_s32)x                                   |
| MLIB_FLOAT  | MLIB_INT    | (mllib_s32)clamp(x, -2147483647-1, 2147483647) |
| MLIB_DOUBLE | MLIB_INT    | (mllib_s32)clamp(x, -2147483647-1, 2147483647) |
| MLIB_BYTE   | MLIB_FLOAT  | (mllib_f32)x                                   |
| MLIB_SHORT  | MLIB_FLOAT  | (mllib_f32)x                                   |
| MLIB_USHORT | MLIB_FLOAT  | (mllib_f32)x                                   |
| MLIB_INT    | MLIB_FLOAT  | (mllib_f32)x                                   |
| MLIB_DOUBLE | MLIB_FLOAT  | (mllib_f32)x                                   |
| MLIB_BYTE   | MLIB_DOUBLE | (mllib_d64)x                                   |
| MLIB_SHORT  | MLIB_DOUBLE | (mllib_d64)x                                   |
| MLIB_USHORT | MLIB_DOUBLE | (mllib_d64)x                                   |
| MLIB_INT    | MLIB_DOUBLE | (mllib_d64)x                                   |
| MLIB_FLOAT  | MLIB_DOUBLE | (mllib_d64)x                                   |

The actions are defined in C-style pseudo-code. All type casts follow the rules of standard C. `clamp()` can be defined as a macro: `#define clamp(x, low, high) (((x) < (low)) ? (low) : (((x) > (high)) ? (high) : (x)))`

**Parameters** The function takes the following arguments:

|                          |                                                                                                        |
|--------------------------|--------------------------------------------------------------------------------------------------------|
| <i>dstData</i>           | The pointer to the destination image data buffer.                                                      |
| <i>srcData</i>           | The pointer to the source image data buffer.                                                           |
| <i>numBands</i>          | The number of channels of the image data buffers.                                                      |
| <i>xSize</i>             | The width of the image.                                                                                |
| <i>ySize</i>             | The height of the image.                                                                               |
| <i>dstDataType</i>       | The data type of the <i>dstData</i> buffer.                                                            |
| <i>dstBandoffsets</i>    | The initial pixel's offsets in the <i>dstData</i> buffer in terms of destination data buffer elements. |
| <i>dstScanlinestride</i> | The scanline stride of the <i>dstData</i> buffer in terms of destination data buffer elements.         |
| <i>dstPixelstride</i>    | The pixel stride of the <i>dstData</i> buffer in terms of destination data buffer elements.            |
| <i>srcDataType</i>       | The data type of the <i>srcData</i> buffer.                                                            |
| <i>srcBandoffsets</i>    | The initial pixel's offsets in the <i>srcData</i> buffer in terms of source data buffer elements.      |
| <i>srcScanlinestride</i> | The scanline stride of the <i>srcData</i> buffer in terms of source data buffer elements.              |
| <i>srcPixelstride</i>    | The pixel stride of the <i>srcData</i> buffer in terms of source data buffer elements.                 |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageDataTypeConvert\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageReplaceColor – replace a color in an image

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageReplaceColor(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *color1, const mllib_s32 *color2);
```

**Description** The `mllib_ImageReplaceColor()` function copies the source image to the destination image and replaces the pixels having a value of *color1* with *color2*.

It uses the following equation:

```
dst[x][y] = color2 if src[x][y] == color1
dst[x][y] = src[x][y] if src[x][y] != color1
```

**Parameters** The function takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to source image.
- color1*        Array of color components to be replaced.
- color2*        Array of color components to replace *color1*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageReplaceColor\\_Inp\(3MLIB\)](#), [mllib\\_ImageReplaceColor\\_Fp\(3MLIB\)](#), [mllib\\_ImageReplaceColor\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageReplaceColor\_Fp – replace a color in an image

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageReplaceColor_Fp(mllib_image *dst,
 const mllib_image *src, const mllib_d64 *color1, const mllib_d64 *color2);
```

**Description** The `mllib_ImageReplaceColor_Fp()` function copies the source image to the destination image and replaces the pixels having a value of *color1* with *color2*.

It uses the following equation:

```
dst[x][y] = color2 if src[x][y] == color1
dst[x][y] = src[x][y] if src[x][y] != color1
```

**Parameters** The function takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to source image.
- color1*        Array of color components to be replaced.
- color2*        Array of color components to replace *color1*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageReplaceColor\(3MLIB\)](#), [mllib\\_ImageReplaceColor\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageReplaceColor\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageReplaceColor\_Fp\_Inp – replace a color in an image, in place

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_ImageReplaceColor_Fp_Inp(mlib_image *srcdst,
 const mlib_d64 *color1, const mlib_d64 *color2);
```

**Description** The `mlib_ImageReplaceColor_Fp_Inp()` function scans the image for all pixels with color value equal to *color1* and replaces these pixels with *color2*.

It uses the following equation:

$$\text{srcdst}[x][y] = \text{color2} \quad \text{if } \text{srcdst}[x][y] == \text{color1}$$

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to the source and destination image.

*color1*     Array of color components to be replaced.

*color2*     Array of color components to replace *color1*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageReplaceColor\(3MLIB\)](#), [mlib\\_ImageReplaceColor\\_Inp\(3MLIB\)](#),  
[mlib\\_ImageReplaceColor\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh5\(3MLIB\)](#),  
[mlib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [mlib\\_ImageThresh5\\_Fp\(3MLIB\)](#),  
[mlib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageReplaceColor\_Inp – replace a color in an image, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageReplaceColor_Inp(mllib_image *srcdst,
 const mlib_s32 *color1, const mlib_s32 *color2);
```

**Description** The `mllib_ImageReplaceColor_Inp()` function scans the image for all pixels with color value equal to *color1* and replaces these pixels with *color2*.

It uses the following equation:

`srcdst[x][y] = color2 if srcdst[x][y] == color1`

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to the source and destination image.

*color1*     Array of color components to be replaced.

*color2*     Array of color components to replace *color1*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageReplaceColor\(3MLIB\)](#), [mllib\\_ImageReplaceColor\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageReplaceColor\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageResetStruct – reset image data structure

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageResetStruct(mllib_image *image, mllib_type type,
 mllib_s32 channels, mllib_s32 width, mllib_s32 height, mllib_s32 stride,
 const void *datbuf);
```

**Description** The `mllib_ImageResetStruct()` function resets a mediaLib image data structure using parameters supplied by the user.

The `mllib_ImageResetStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- `image` should not be `NULL`
- `type` should be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`
- `channels` should be between 1 and 4
- `width` should be greater than 0
- `height` should be greater than 0
- `stride` should be no less than `width * channels * (size of type in bytes)`

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

When `datbuf` is `NULL`, the original data buffer is reused. If `mllib_ImageIsUserAllocated(image) == 0`, such as the case the image data structure was created by `mllib_ImageCreate()`, and the data buffer size required by the parameters supplied is larger than the original, `MLIB_FAILURE` is returned.

When `datbuf` is not `NULL`, if `mllib_ImageIsUserAllocated(image) == 0`, the original data buffer is freed, otherwise the original data buffer is not freed. If `datbuf` points to the original data buffer, it is not freed.

**Parameters** The function takes the following arguments:

|                 |                                                                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>image</i>    | Pointer to the image data structure.                                                                                                                                                                             |
| <i>type</i>     | Image data type. It can be <code>MLIB_BIT</code> , <code>MLIB_BYTE</code> , <code>MLIB_SHORT</code> , <code>MLIB_USHORT</code> , <code>MLIB_INT</code> , <code>MLIB_FLOAT</code> , or <code>MLIB_DOUBLE</code> . |
| <i>channels</i> | Number of channels in the image.                                                                                                                                                                                 |
| <i>width</i>    | Width of image in pixels.                                                                                                                                                                                        |
| <i>height</i>   | Height of image in pixels.                                                                                                                                                                                       |
| <i>stride</i>   | Stride of each row of the data space in bytes.                                                                                                                                                                   |
| <i>datbuf</i>   | Pointer to the image data buffer.                                                                                                                                                                                |

**Return Values** MLIB\_SUCCESS is returned if the image data structure is reset successfully. MLIB\_FAILURE is returned when the image data structure can not be reset according to the parameters supplied.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageCreate\(3MLIB\)](#), [mllib\\_ImageCreateSubimage\(3MLIB\)](#),  
[mllib\\_ImageCreateStruct\(3MLIB\)](#), [mllib\\_ImageSetStruct\(3MLIB\)](#),  
[mllib\\_ImageDelete\(3MLIB\)](#), [mllib\\_ImageSetFormat\(3MLIB\)](#),  
[mllib\\_ImageSetPaddings\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageResetSubimageStruct – reset sub-image data structure

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageResetSubimageStruct(mllib_image *subimg,
 const mllib_image *img, mllib_s32 x, mllib_s32 y,
 mllib_s32 w, mllib_s32 h);
```

**Description** The `mllib_ImageResetSubimageStruct()` function resets a sub-image's data structure using parameters supplied by the user.

The `mllib_ImageResetSubimageStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- `subimg != NULL`
- `img != NULL`
- $0 < w \leq \text{mllib\_ImageGetWidth}(img)$
- $0 < h \leq \text{mllib\_ImageGetHeight}(img)$
- $0 \leq x \leq (\text{mllib\_ImageGetWidth}(img) - w)$
- $0 \leq y \leq (\text{mllib\_ImageGetHeight}(img) - h)$

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

If `mllib_ImageIsUserAllocated(subimg) == 0`, the original data buffer is freed, otherwise the original data buffer is not freed.

**Parameters** The function takes the following arguments:

*subimg*     Pointer to the sub-image data structure.

*img*        Pointer to the source image data structure.

*x*          X coordinate of the left border in the source image.

*y*          Y coordinate of the top border in the source image.

*w*          Width of the sub-image.

*h*          Height of the sub-image.

**Return Values** `MLIB_SUCCESS` is returned if the image data structure is reset successfully. `MLIB_FAILURE` is returned when the image data structure can not be reset according to the parameters supplied.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageCreate(3MLIB)`, `mllib_ImageCreateSubimage(3MLIB)`,  
`mllib_ImageCreateStruct(3MLIB)`, `mllib_ImageSetStruct(3MLIB)`,  
`mllib_ImageResetStruct(3MLIB)`, `mllib_ImageSetSubimageStruct(3MLIB)`,  
`mllib_ImageDelete(3MLIB)`, `mllib_ImageSetFormat(3MLIB)`,  
`mllib_ImageSetPaddings(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageRotate180 – rotate an image by 180 degrees

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageRotate180(mllib_image *dst, const mllib_image *src);
```

**Description** Rotate an image 180 degrees.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be MLIB\_BIT, MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, or MLIB\_INT.

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mllib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipX\(3MLIB\)](#), [mllib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipY\(3MLIB\)](#), [mllib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate90\(3MLIB\)](#), [mllib\\_ImageRotate90\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate180\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate270\(3MLIB\)](#), [mllib\\_ImageRotate270\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRotate180\_Fp – rotate an image by 180 degrees

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageRotate180_Fp(mllib_image *dst, const mllib_image *src);`

**Description** Rotate a floating-point image 180 degrees.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mllib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipMainDiag\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipX\(3MLIB\)](#), [mllib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipY\(3MLIB\)](#),  
[mllib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate90\(3MLIB\)](#),  
[mllib\\_ImageRotate90\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate180\(3MLIB\)](#),  
[mllib\\_ImageRotate270\(3MLIB\)](#), [mllib\\_ImageRotate270\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageRotate270 – rotate an image by 270 degrees

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_ImageRotate270(mlib_image *dst, const mlib_image *src);
```

**Description** Rotate an image 270 degrees clockwise.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be MLIB\_BIT, MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, or MLIB\_INT.

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mlib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#), [mlib\\_ImageFlipMainDiag\(3MLIB\)](#), [mlib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#), [mlib\\_ImageFlipX\(3MLIB\)](#), [mlib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mlib\\_ImageFlipY\(3MLIB\)](#), [mlib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mlib\\_ImageRotate90\(3MLIB\)](#), [mlib\\_ImageRotate90\\_Fp\(3MLIB\)](#), [mlib\\_ImageRotate180\(3MLIB\)](#), [mlib\\_ImageRotate180\\_Fp\(3MLIB\)](#), [mlib\\_ImageRotate270\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRotate270\_Fp – rotate an image by 270 degrees

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageRotate270_Fp(mllib_image *dst, const mllib_image *src);`

**Description** Rotate a floating-point image 270 degrees clockwise.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mllib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipMainDiag\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipX\(3MLIB\)](#), [mllib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipY\(3MLIB\)](#),  
[mllib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate90\(3MLIB\)](#),  
[mllib\\_ImageRotate90\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate180\(3MLIB\)](#),  
[mllib\\_ImageRotate180\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate270\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageRotate – rotate image

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageRotate(mllib_image *dst, const mllib_image *src,
 mllib_d64 angle, mllib_d64 xcenter, mllib_d64 ycenter, mllib_filter filter,
 mllib_edge edge);
```

**Description** The `mllib_ImageRotate()` function rotates a source image around a user-defined rotation center in the user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The (`xcenter`, `ycenter`) point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

**Parameters** The function takes the following arguments:

|                |                                                                                                                                                                                                                                                       |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>     | Pointer to destination image.                                                                                                                                                                                                                         |
| <i>src</i>     | Pointer to source image.                                                                                                                                                                                                                              |
| <i>angle</i>   | Angle of rotation. The angle is measured in radians clockwise.                                                                                                                                                                                        |
| <i>xcenter</i> | X coordinate of rotation center in the source image.                                                                                                                                                                                                  |
| <i>ycenter</i> | Y coordinate of rotation center in the source image.                                                                                                                                                                                                  |
| <i>filter</i>  | Type of resampling filter. It can be one of the following:<br><code>MLIB_NEAREST</code><br><code>MLIB_BILINEAR</code><br><code>MLIB_BICUBIC</code><br><code>MLIB_BICUBIC2</code>                                                                      |
| <i>edge</i>    | Type of edge condition. It can be one of the following:<br><code>MLIB_EDGE_DST_NO_WRITE</code><br><code>MLIB_EDGE_DST_FILL_ZERO</code><br><code>MLIB_EDGE_OP_NEAREST</code><br><code>MLIB_EDGE_SRC_EXTEND</code><br><code>MLIB_EDGE_SRC_PADDED</code> |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageRotate\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotateIndex\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRotate90 – rotate an image by 90 degrees

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageRotate90(mllib_image *dst, const mllib_image *src);
```

**Description** Rotate an image 90 degrees clockwise.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be MLIB\_BIT, MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, or MLIB\_INT.

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mllib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipX\(3MLIB\)](#), [mllib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipY\(3MLIB\)](#), [mllib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate90\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate180\(3MLIB\)](#), [mllib\\_ImageRotate180\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate270\(3MLIB\)](#), [mllib\\_ImageRotate270\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRotate90\_Fp – rotate an image by 90 degrees

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageRotate90_Fp(mllib_image *dst, const mllib_image *src);`

**Description** Rotate a floating-point image 90 degrees clockwise.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to source image.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_ImageFlipAntiDiag\(3MLIB\)](#), [mllib\\_ImageFlipAntiDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipMainDiag\(3MLIB\)](#), [mllib\\_ImageFlipMainDiag\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageFlipX\(3MLIB\)](#), [mllib\\_ImageFlipX\\_Fp\(3MLIB\)](#), [mllib\\_ImageFlipY\(3MLIB\)](#),  
[mllib\\_ImageFlipY\\_Fp\(3MLIB\)](#), [mllib\\_ImageRotate90\(3MLIB\)](#),  
[mllib\\_ImageRotate180\(3MLIB\)](#), [mllib\\_ImageRotate180\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageRotate270\(3MLIB\)](#), [mllib\\_ImageRotate270\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageRotate\_Fp – rotate image

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageRotate_Fp(mllib_image *dst, const mllib_image *src,
 mllib_d64 angle, mllib_d64 xcenter, mllib_d64 ycenter, mllib_filter filter,
 mllib_edge edge);
```

**Description** The `mllib_ImageRotate_Fp()` function rotates a floating-point source image around a user-defined rotation center in the user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The `(xcenter, ycenter)` point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The center of the upper-left corner pixel of an image is located at `(0.5, 0.5)`.

**Parameters** The function takes the following arguments:

|                |                                                                                                                                                                                          |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>     | Pointer to destination image.                                                                                                                                                            |
| <i>src</i>     | Pointer to source image.                                                                                                                                                                 |
| <i>angle</i>   | Angle of rotation. The angle is measured in radians clockwise.                                                                                                                           |
| <i>xcenter</i> | X coordinate of rotation center in the source image.                                                                                                                                     |
| <i>ycenter</i> | Y coordinate of rotation center in the source image.                                                                                                                                     |
| <i>filter</i>  | Type of resampling filter. It can be one of the following:<br><br>MLIB_NEAREST<br>MLIB_BILINEAR<br>MLIB_BICUBIC<br>MLIB_BICUBIC2                                                         |
| <i>edge</i>    | Type of edge condition. It can be one of the following:<br><br>MLIB_EDGE_DST_NO_WRITE<br>MLIB_EDGE_DST_FILL_ZERO<br>MLIB_EDGE_OP_NEAREST<br>MLIB_EDGE_SRC_EXTEND<br>MLIB_EDGE_SRC_PADDED |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageRotate\(3MLIB\)](#), [mllib\\_ImageRotateIndex\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageRotateIndex – rotate color-indexed image

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_ImageRotateIndex(mlib_image *dst, const mlib_image *src,
 mlib_d64 angle, mlib_d64 xcenter, mlib_d64 ycenter, mlib_filter filter,
 mlib_edge edge, const void *colormap);
```

**Description** The `mlib_ImageRotateIndex()` function rotates the source image about a user-defined rotation center in user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The (`xcenter`, `ycenter`) point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The source and destination images must be single-channel images.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

**Parameters** The function takes the following arguments:

|                 |                                                                                                                                                                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>      | Pointer to destination image.                                                                                                                                                                                                                         |
| <i>src</i>      | Pointer to source image.                                                                                                                                                                                                                              |
| <i>angle</i>    | Angle of rotation. The angle is measured in radians clockwise.                                                                                                                                                                                        |
| <i>xcenter</i>  | X coordinate of rotation center in the source image.                                                                                                                                                                                                  |
| <i>ycenter</i>  | Y coordinate of rotation center in the source image.                                                                                                                                                                                                  |
| <i>filter</i>   | Type of resampling filter. It can be one of the following:<br><code>MLIB_NEAREST</code><br><code>MLIB_BILINEAR</code><br><code>MLIB_BICUBIC</code><br><code>MLIB_BICUBIC2</code>                                                                      |
| <i>edge</i>     | Type of edge condition. It can be one of the following:<br><code>MLIB_EDGE_DST_NO_WRITE</code><br><code>MLIB_EDGE_DST_FILL_ZERO</code><br><code>MLIB_EDGE_OP_NEAREST</code><br><code>MLIB_EDGE_SRC_EXTEND</code><br><code>MLIB_EDGE_SRC_PADDED</code> |
| <i>colormap</i> | Internal data structure for inverse color mapping. This data structure is generated by the <code>mlib_ImageColorTrue2IndexInit()</code> function.                                                                                                     |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageRotate\(3MLIB\)](#), [mllib\\_ImageRotate\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageScalarBlend – image blending with scalar

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageScalarBlend(mllib_image dst, const mllib_image *src1,
 const mllib_image *src2, const mllib_s32 *alpha);
```

**Description** The `mllib_ImageScalarBlend()` function blends the first and second source images by adding each of their scaled pixels. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by  $(1 - a)$ .

It uses the following equation:

$$dst[x][y][i] = a[i]*src1[x][y][i] + (1 - a[i])*src2[x][y][i]$$

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src1* Pointer to first source image.

*src2* Pointer to second source image.

*alpha* Scalar blending factor. The `a` value equals  $(\alpha * 2^{**(-31)})$ . `alpha[i]` contains the blending factor for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScalarBlend\\_Fp\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScalarBlend\_Fp – image blending with scalar

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageScalarBlend_Fp(mllib_image *dst,
 const mllib_image *src1, const mllib_image *src2, const mllib_d64 *alpha);
```

**Description** The `mllib_ImageScalarBlend_Fp()` function blends the first and second floating-point source images by adding each of their scaled pixels. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by  $(1 - a)$ .

It uses the following equation:

$$dst[x][y][i] = a[i]*src1[x][y][i] + (1 - a[i])*src2[x][y][i]$$

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src1* Pointer to first source image.
- src2* Pointer to second source image.
- alpha* Scalar blending factor. The `a` value equals `alpha` which should be in the  $[0.0, 1.0]$  range. `alpha[i]` contains the blending factor for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScalarBlend\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScalarBlend\_Fp\_Inp – image blending with scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageScalarBlend_Fp_Inp(mllib_image *src1dst,
 const mllib_image *src2, const mllib_d64 *alpha);
```

**Description** The `mllib_ImageScalarBlend_Fp_Inp()` function blends the first and second floating-point source images by adding each of their scaled pixels in place. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by  $(1 - a)$ .

It uses the following equation:

$$\text{src1dst}[x][y][i] = a[i] * \text{src1dst}[x][y][i] + (1 - a[i]) * \text{src2}[x][y][i]$$

**Parameters** The function takes the following arguments:

- src1dst*     Pointer to first source and destination image.
- src2*        Pointer to second source image.
- alpha*       Scalar blending factor. The `a` value equals `alpha` which should be in the  $[0.0, 1.0]$  range. `alpha[i]` contains the blending factor for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScalarBlend\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Fp\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScalarBlend\_Inp – image blending with scalar, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageScalarBlend_Inp(mllib_image *src1dst,
 const mllib_image *src2, const mllib_s32 *alpha);
```

**Description** The `mllib_ImageScalarBlend_Inp()` function blends the first and second source images by adding each of their scaled pixels in place. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by  $(1 - a)$ .

It uses the following equation:

$$\text{src1dst}[x][y][i] = a[i] * \text{src1dst}[x][y][i] + (1 - a[i]) * \text{src2}[x][y][i]$$

**Parameters** The function takes the following arguments:

- src1dst*     Pointer to first source and destination image.
- src2*        Pointer to second source image.
- alpha*       Scalar blending factor. The `a` value equals  $(\text{alpha} * 2^{**(-31)})$ . `alpha[i]` contains the blending factor for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScalarBlend\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Fp\(3MLIB\)](#), [mllib\\_ImageScalarBlend\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScale2 – linear scaling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageScale2(mllib_image *dst, const mllib_image *src,
 const mllib_d64 *alpha, const mllib_d64 *beta);
```

**Description** The `mllib_ImageScale2()` function performs a linear scaling on the pixels of the source image by multiplying the data by a scale factor and then adding an offset. Images must have the same size, and number of channels. They can have 1, 2, 3, or 4 channels.

The following equation is used:

$$dst[x][y][i] = src[x][y][i] * alpha[i] + beta[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

| Type[*]     | BYTE | SHORT | USHORT | INT | FLOAT | DOUBLE |
|-------------|------|-------|--------|-----|-------|--------|
| MLIB_BYTE   | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_SHORT  | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_USHORT | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_INT    | Y    | Y     | Y      | Y   | Y     | Y      |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*alpha* Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.

*beta* Offset value. `beta[i]` contains the offset for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScale\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageScale\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageScale\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageScale2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScale2\_Inp – linear scaling, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageScale2_Inp(mllib_image *srcdst, const mllib_d64 *alpha,
 const mllib_d64 *beta);
```

**Description** The `mllib_ImageScale2_Inp()` function performs an in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor and then adding an offset. Images can have 1, 2, 3, or 4 channels.

The following equation is used:

$$\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] * \alpha[i] + \beta[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT` or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to source and destination image.

*alpha*     Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.

*beta*       Offset value. `beta[i]` contains the offset for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScale\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageScale\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageScale\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageScale2\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScale – linear scaling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageScale(mllib_image *dst,
 const mllib_image *src, const mllib_s32 *alpha, const mllib_s32 *beta,
 mllib_s32 shift);
```

**Description** The `mllib_ImageScale()` function performs a linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

$$dst[x][y][i] = src[x][y][i] * alpha[i] * 2^{*(-shift)} + beta[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

| Type[*]     | BYTE | SHORT | USHORT | INT | FLOAT | DOUBLE |
|-------------|------|-------|--------|-----|-------|--------|
| MLIB_BYTE   | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_SHORT  | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_USHORT | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_INT    | Y    | Y     | Y      | Y   | Y     | Y      |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- alpha* Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- beta* Offset value. `beta[i]` contains the offset for channel `i`.
- shift* Right shifting factor.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.



**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScale\\_Fp\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageScale\\_Inp\(3MLIB\)](#), [mllib\\_ImageScale2\(3MLIB\)](#),  
[mllib\\_ImageScale2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScale\_Fp – linear scaling

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageScale_Fp(mllib_image *dst, const mllib_image *src,
 const mlib_d64 *alpha, const mlib_d64 *beta);
```

**Description** The `mllib_ImageScale_Fp()` function performs a floating-point linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

$$dst[x][y][i] = src[x][y][i] * alpha[i] + beta[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

| Type[*]     | BYTE | SHORT | USHORT | INT | FLOAT | DOUBLE |
|-------------|------|-------|--------|-----|-------|--------|
| MLIB_FLOAT  | Y    | Y     | Y      | Y   | Y     | Y      |
| MLIB_DOUBLE | Y    | Y     | Y      | Y   | Y     | Y      |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- alpha*      Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- beta*       Offset value. `beta[i]` contains the offset for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageScale(3MLIB)`, `mlib_ImageScale_Fp_Inp(3MLIB)`,  
`mlib_ImageScale_Inp(3MLIB)`, `mlib_ImageScale2(3MLIB)`,  
`mlib_ImageScale2_Inp(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageScale\_Fp\_Inp – linear scaling, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageScale_Fp_Inp(mllib_image *srcdst,
 const mllib_d64 *alpha, const mllib_d64 *beta);
```

**Description** The `mllib_ImageScale_Fp_Inp()` function performs a floating-point, in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

$$\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] * \text{alpha}[i] + \text{beta}[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type `MLIB_FLOAT` or `MLIB_DOUBLE`.

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- alpha*     Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- beta*       Offset value. `beta[i]` contains the offset for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageScale\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\(3MLIB\)](#), [mllib\\_ImageScale\\_Inp\(3MLIB\)](#), [mllib\\_ImageScale2\(3MLIB\)](#), [mllib\\_ImageScale2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageScale\_Inp – linear scaling, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageScale_Inp(mllib_image *srcdst, const mllib_s32 *alpha,
 const mllib_s32 *beta, mllib_s32 shift);
```

**Description** The `mllib_ImageScale_Inp()` function performs an in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

$$\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] * \text{alpha}[i] * 2^{*(-\text{shift})} + \text{beta}[i]$$

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT` or `MLIB_INT`.

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- alpha*     Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- beta*       Offset value. `beta[i]` contains the offset for channel `i`.
- shift*      Right shifting factor.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_ImageScale\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\(3MLIB\)](#), [mllib\\_ImageScale\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageScale2\(3MLIB\)](#), [mllib\\_ImageScale2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSConv3x3 – separable 3x3 convolution

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSConv3x3(mllib_image *dst, const mllib_image *src,
 const mlib_s32 *hkernel, const mlib_s32 *vkernel, mlib_s32 scale,
 mlib_s32 cmask, mlib_edge edge);
```

**Description** The `mllib_ImageSConv3x3()` function performs a separable 3x3 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q] * 2^{**(-2*scale)}$$

where  $m = 3$ ,  $n = 3$ ,  $dm = (m - 1)/2 = 1$ ,  $dn = (n - 1)/2 = 1$ .

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- scale* Scaling factor.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3_Fp(3MLIB)`,  
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,  
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConv3x3\_Fp – separable 3x3 convolution

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSConv3x3_Fp(mllib_image *dst, const mllib_image *src,
 const mlib_d64 *hkernel, const mlib_d64 *vkernel, mlib_s32 cmask,
 mlib_edge edge);
```

**Description** The mllib\_ImageSConv3x3\_Fp() function performs a separable 3x3 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q]$$

where m = 3, n = 3, dm = (m - 1)/2 = 1, dn = (n - 1)/2 = 1.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3(3MLIB)`,  
`mllib_ImageSConv5x5(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,  
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConv5x5 – separable 5x5 convolution

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSConv5x5(mllib_image *dst, const mllib_image *src,
 const mlib_s32 *hkernel, const mlib_s32 *vkernel, mlib_s32 scale,
 mlib_s32 cmask, mlib_edge edge);
```

**Description** The `mllib_ImageSConv5x5()` function performs a separable 5x5 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q] * 2^{**(-2*scale)}$$

where  $m = 5$ ,  $n = 5$ ,  $dm = (m - 1)/2 = 2$ ,  $dn = (n - 1)/2 = 2$ .

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- scale* Scaling factor.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3(3MLIB)`,  
`mllib_ImageSConv3x3_Fp(3MLIB)`, `mllib_ImageSConv5x5_Fp(3MLIB)`,  
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConv5x5\_Fp – separable 5x5 convolution

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSConv5x5_Fp(mllib_image *dst, const mllib_image *src,
 const mlib_d64 *hkernel, const mlib_d64 *vkernel, mlib_s32 cmask,
 mlib_edge edge);
```

**Description** The `mllib_ImageSConv5x5_Fp()` function performs a separable 5x5 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q]$$

where  $m = 5$ ,  $n = 5$ ,  $dm = (m - 1)/2 = 2$ ,  $dn = (n - 1)/2 = 2$ .

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
- MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3(3MLIB)`,  
`mllib_ImageSConv3x3_Fp(3MLIB)`, `mllib_ImageSConv5x5(3MLIB)`,  
`mllib_ImageSConv7x7(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConv7x7 – separable 7x7 convolution

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSConv7x7(mllib_image *dst, const mllib_image *src,
 const mlib_s32 *hkernel, const mlib_s32 *vkernel, mlib_s32 scale,
 mlib_s32 cmask, mlib_edge edge);
```

**Description** The `mllib_ImageSConv7x7()` function performs a separable 7x7 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q] * 2^{*(-2*scale)}$$

where  $m = 7$ ,  $n = 7$ ,  $dm = (m - 1)/2 = 3$ ,  $dn = (n - 1)/2 = 3$ .

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- scale* Scaling factor.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3(3MLIB)`,  
`mllib_ImageSConv3x3_Fp(3MLIB)`, `mllib_ImageSConv5x5(3MLIB)`,  
`mllib_ImageSConv5x5_Fp(3MLIB)`, `mllib_ImageSConv7x7_Fp(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConv7x7\_Fp – separable 7x7 convolution

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSConv7x7_Fp(mllib_image *dst, const mllib_image *src,
 const mlib_d64 *hkernel, const mlib_d64 *vkernel, mlib_s32 cmask,
 mlib_edge edge);
```

**Description** The `mllib_ImageSConv7x7_Fp()` function performs a separable 7x7 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * h[p] * v[q]$$

where  $m = 7$ ,  $n = 7$ ,  $dm = (m - 1)/2 = 3$ ,  $dn = (n - 1)/2 = 3$ .

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- hkernel* Pointer to the horizontal kernel.
- vkernel* Pointer to the vertical kernel.
- cmask* Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SRC\_EXTEND

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



---

**See Also** `mllib_ImageConv2x2(3MLIB)`, `mllib_ImageConv2x2_Fp(3MLIB)`,  
`mllib_ImageConv2x2Index(3MLIB)`, `mllib_ImageConv3x3(3MLIB)`,  
`mllib_ImageConv3x3_Fp(3MLIB)`, `mllib_ImageConv3x3Index(3MLIB)`,  
`mllib_ImageConv4x4(3MLIB)`, `mllib_ImageConv4x4_Fp(3MLIB)`,  
`mllib_ImageConv4x4Index(3MLIB)`, `mllib_ImageConv5x5(3MLIB)`,  
`mllib_ImageConv5x5_Fp(3MLIB)`, `mllib_ImageConv5x5Index(3MLIB)`,  
`mllib_ImageConv7x7(3MLIB)`, `mllib_ImageConv7x7_Fp(3MLIB)`,  
`mllib_ImageConv7x7Index(3MLIB)`, `mllib_ImageConvKernelConvert(3MLIB)`,  
`mllib_ImageConvMxN(3MLIB)`, `mllib_ImageConvMxN_Fp(3MLIB)`,  
`mllib_ImageConvMxNIndex(3MLIB)`, `mllib_ImageConvolveMxN(3MLIB)`,  
`mllib_ImageConvolveMxN_Fp(3MLIB)`, `mllib_ImageSConv3x3(3MLIB)`,  
`mllib_ImageSConv3x3_Fp(3MLIB)`, `mllib_ImageSConv5x5(3MLIB)`,  
`mllib_ImageSConv5x5_Fp(3MLIB)`, `mllib_ImageSConv7x7(3MLIB)`,  
`mllib_ImageSConvKernelConvert(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSConvKernelConvert – kernel conversion for separable convolution

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSConvKernelConvert(mllib_s32 *ihkernel,
 mllib_s32 *ivkernel, mllib_s32 *iscale, const mllib_d64 *fhkernel,
 const mllib_d64 *fvkernel, mllib_s32 m, mllib_s32 n, mllib_type type);
```

**Description** The `mllib_ImageSConvKernelConvert()` function converts a floating-point separable convolution kernel to an integer kernel with its scaling factor, which is suitable to be used in separable convolution functions.

**Parameters** The function takes the following arguments:

- ihkernel* Pointer to integer horizontal kernel.
- ivkernel* Pointer to integer vertical kernel.
- iscale* Scaling factor of the integer convolution kernel.
- fhkernel* Pointer to floating-point horizontal kernel.
- fvkernel* Pointer to floating-point vertical kernel.
- m* Width of the convolution kernel. *m* must be an odd number larger than 1.
- n* Height of the convolution kernel. *n* must be an odd number larger than 1.
- type* The image type.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSConv3x3\(3MLIB\)](#), [mllib\\_ImageSConv3x3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageSConv5x5\(3MLIB\)](#), [mllib\\_ImageSConv5x5\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageSConv7x7\(3MLIB\)](#), [mllib\\_ImageSConv7x7\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageConvKernelConvert\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSetFormat – set format

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageSetFormat(mllib_image *img, mllib_format format);`

**Description** The `mllib_ImageSetFormat()` function sets a new value for the `format` field of a `mllib_image` structure.

The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

**Parameters** The function takes the following arguments:

*img* Pointer to a mediaLib image structure.

*format* Image pixel format. It can be one of the following:

`MLIB_FORMAT_UNKNOWN`  
`MLIB_FORMAT_INDEXED`  
`MLIB_FORMAT_GRAYSCALE`  
`MLIB_FORMAT_RGB`  
`MLIB_FORMAT_BGR`  
`MLIB_FORMAT_ARGB`  
`MLIB_FORMAT_ABGR`  
`MLIB_FORMAT_PACKED_ARGB`  
`MLIB_FORMAT_PACKED_ABGR`  
`MLIB_FORMAT_GRAYSCALE_ALPHA`  
`MLIB_FORMAT_RGBA`

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageGetFormat\(3MLIB\)](#), [mllib\\_ImageCreate\(3MLIB\)](#),  
[mllib\\_ImageCreateStruct\(3MLIB\)](#), [mllib\\_ImageCreateSubimage\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSetPaddings – set paddings

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSetPaddings(mllib_image *img, mllib_u8 left,
 mllib_u8 top, mllib_u8 right, mllib_u8 bottom);
```

**Description** The `mllib_ImageSetPaddings()` function sets new values for the paddings field of the `mllib_image` structure as follows:

```
img->paddings[0] = left;
img->paddings[1] = top;
img->paddings[2] = right;
img->paddings[3] = bottom;
```

By default, an image structure creation function, such as `mllib_ImageCreate()`, `mllib_ImageCreateStruct()`, or `mllib_ImageCreateSubimage()`, sets the paddings field of the `mllib_image` structure as follows:

```
img->paddings[0] = 0;
img->paddings[1] = 0;
img->paddings[2] = 0;
img->paddings[3] = 0;
```

Note that this function is needed only when the edge condition `MLIB_EDGE_SRC_PADDED` is used.

The `mllib_image->paddings` field denotes the amount of paddings on each side of an image, from which the real image border can be seen. When `MLIB_EDGE_SRC_PADDED` is specified as the edge condition, a geometric function uses the "real" source image border for clipping the destination image.

**Parameters** The function takes the following arguments:

|               |                                             |
|---------------|---------------------------------------------|
| <i>img</i>    | Pointer to image data structure.            |
| <i>left</i>   | Number of columns padded on the left side.  |
| <i>top</i>    | Number of rows padded on the top.           |
| <i>right</i>  | Number of columns padded on the right side. |
| <i>bottom</i> | Number of rows padded at the bottom.        |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageGetPaddings\(3MLIB\)](#), [mlib\\_ImageCreate\(3MLIB\)](#),  
[mlib\\_ImageCreateStruct\(3MLIB\)](#), [mlib\\_ImageCreateSubimage\(3MLIB\)](#),  
[mlib\\_ImageAffine\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSetStruct – set image data structure

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSetStruct(mllib_image *image, mllib_type type,
 mllib_s32 channels, mllib_s32 width, mllib_s32 height, mllib_s32 stride,
 const void *datbuf);
```

**Description** The `mllib_ImageSetStruct()` function sets a mediaLib image data structure using parameters supplied by the user.

The `mllib_ImageSetStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- `image` should not be `NULL`
- `type` should be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`
- `channels` should be between 1 and 4
- `width` should be greater than 0
- `height` should be greater than 0
- `stride` should be no less than `width * channels * (size of type in bytes)`
- `datbuf` should not be `NULL`

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

If the data buffer in the image data structure is not `NULL`, it is the user's responsibility to free it if necessary.

**Parameters** The function takes the following arguments:

|                 |                                                                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>image</i>    | Pointer to the image data structure.                                                                                                                                                                             |
| <i>type</i>     | Image data type. It can be <code>MLIB_BIT</code> , <code>MLIB_BYTE</code> , <code>MLIB_SHORT</code> , <code>MLIB_USHORT</code> , <code>MLIB_INT</code> , <code>MLIB_FLOAT</code> , or <code>MLIB_DOUBLE</code> . |
| <i>channels</i> | Number of channels in the image.                                                                                                                                                                                 |
| <i>width</i>    | Width of image in pixels.                                                                                                                                                                                        |
| <i>height</i>   | Height of image in pixels.                                                                                                                                                                                       |
| <i>stride</i>   | Stride of each row of the data space in bytes.                                                                                                                                                                   |
| <i>datbuf</i>   | Pointer to the image data buffer.                                                                                                                                                                                |

**Return Values** MLIB\_SUCCESS is returned if the image data structure is set successfully. MLIB\_FAILURE is returned when the image data structure can not be set according to the parameters supplied.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageCreate\(3MLIB\)](#), [mlib\\_ImageCreateSubimage\(3MLIB\)](#),  
[mlib\\_ImageCreateStruct\(3MLIB\)](#), [mlib\\_ImageResetStruct\(3MLIB\)](#),  
[mlib\\_ImageDelete\(3MLIB\)](#), [mlib\\_ImageSetFormat\(3MLIB\)](#),  
[mlib\\_ImageSetPaddings\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSetSubimageStruct – set sub-image data structure

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSetSubimageStruct(mllib_image *subimg,
 const mllib_image *img, mllib_s32 x, mllib_s32 y,
 mllib_s32 w, mllib_s32 h);
```

**Description** The `mllib_ImageSetSubimageStruct()` function sets a sub-image's data structure using parameters supplied by the user.

The `mllib_ImageSetSubimageStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- `subimg != NULL`
- `img != NULL`
- $0 < w \leq \text{mllib\_ImageGetWidth}(img)$
- $0 < h \leq \text{mllib\_ImageGetHeight}(img)$
- $0 \leq x \leq (\text{mllib\_ImageGetWidth}(img) - w)$
- $0 \leq y \leq (\text{mllib\_ImageGetHeight}(img) - h)$

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

**Parameters** The function takes the following arguments:

- subimg*      Pointer to the sub-image data structure.
- img*          Pointer to the source image data structure.
- x*            X coordinate of the left border in the source image.
- y*            Y coordinate of the top border in the source image.
- w*            Width of the sub-image.
- h*            Height of the sub-image.

**Return Values** `MLIB_SUCCESS` is returned if the image data structure is set successfully. `MLIB_FAILURE` is returned when the image data structure can not be set according to the parameters supplied.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** `mlib_ImageCreate(3MLIB)`, `mlib_ImageCreateSubimage(3MLIB)`,  
`mlib_ImageCreateStruct(3MLIB)`, `mlib_ImageSetStruct(3MLIB)`,  
`mlib_ImageResetStruct(3MLIB)`, `mlib_ImageResetSubimageStruct(3MLIB)`,  
`mlib_ImageDelete(3MLIB)`, `mlib_ImageSetFormat(3MLIB)`,  
`mlib_ImageSetPaddings(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageSobel, mllib\_ImageSobel\_Fp – Sobel filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSobel(mllib_image *dst, const mllib_image *src,
 mllib_s32 cmask, mllib_edge edge);

mllib_status mllib_ImageSobel_Fp(mllib_image *dst, const mllib_image *src,
 mllib_s32 cmask, mllib_edge edge);
```

**Description** Each function is a special case of the gradient filter, which is an edge detector which computes the magnitude of the image gradient vector in two orthogonal directions. In this case, the gradient filter uses specific horizontal and vertical masks.

The Sobel filter is one of the special cases of gradient filter using the following horizontal and vertical masks:

```
hmask = { -1.0, 0.0, 1.0,
 -2.0, 0.0, 2.0,
 -1.0, 0.0, 1.0 }

vmask = { -1.0, -2.0, -1.0,
 0.0, 0.0, 0.0,
 1.0, 2.0, 1.0 }
```

**Parameters** Each function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to source image.
- cmask* Channel mask to indicate the channels to be processed, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DS\_FILL\_ZERO
  - MLIB\_EDGE\_DST\_COPY\_SRC
  - MLIB\_EDGE\_SR\_EXTEND

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mlib\\_ImageGradient3x3\(3MLIB\)](#), [mlib\\_ImageGradient3x3\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSqr\_Fp – square

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSqr_Fp(mllib_image *dst, const mllib_image *src);
```

**Description** The `mllib_ImageSqr_Fp()` function computes the floating-point square of each pixel in the source image.

It uses the following equation:

$$\text{dst}[x][y][i] = \text{src}[x][y][i] * \text{src}[x][y][i]$$

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination image.

*src*     Pointer to source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSqr\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSqr\_Fp\_Inp – square, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSqr_Fp_Inp(mllib_image *srcdst);
```

**Description** The `mllib_ImageSqr_Fp_Inp()` function computes the floating-point square of each pixel in the source image.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] * \text{srcdst}[x][y][i]$$

**Parameters** The function takes the following arguments:

*srcdst*      Pointer to source and destination image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSqr\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSqrShift – square with shifting

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSqrShift(mllib_image *dst, const mllib_image *src,
 mllib_s32 shift);
```

**Description** The `mllib_ImageSqrShift()` function computes the square of each pixel in the source image and scales the result by the shift factor.

It uses the following equation:

$$dst[x][y][i] = src[x][y][i] * src[x][y][i] * 2^{*(-shift)}$$

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src*      Pointer to source image.
- shift*    Right shifting factor.  $0 \leq shift \leq 31$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSqrShift\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSqrShift\_Inp – square with shifting, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSqrShift_Inp(mllib_image *srcdst, mllib_s32 shift);
```

**Description** The `mllib_ImageSqrShift_Inp()` function computes the square of each pixel in the source image and scales the result by the shift factor, in place.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] * \text{srcdst}[x][y][i] * 2^{*(-\text{shift})}$$

**Parameters** The function takes the following arguments:

*srcdst*      Pointer to source and destination image.

*shift*      Right shifting factor.  $0 \leq \text{shift} \leq 31$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSqrShift\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageStdDev – image standard deviation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageStdDev(mllib_d64 *sdev, const mllib_image *img,
 const mllib_d64 *mean);
```

**Description** The `mllib_ImageStdDev()` function computes the standard deviation for each channel in the source image.

It uses the following equation:

$$sdev[i] = \sqrt{\frac{1}{w \cdot h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (img[x][y][i] - mean[i])^2}}$$

where, in the case of `mean == NULL`,

$$mean[i] = \frac{1}{w \cdot h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} img[x][y][i]$$

**Parameters** The function takes the following arguments:

*sdev* Pointer to standard deviation array, whose size is the number of channels in the source image. `sdev[i]` contains the standard deviation of channel `i`.

*img* Pointer to input image.

*mean* Pointer to pre-computed mean array for each channel. (If `NULL`, it will be computed.) `mean[i]` contains the mean of channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMean\(3MLIB\)](#), [mllib\\_ImageMean\\_Fp\(3MLIB\)](#), [mllib\\_ImageStdDev\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageStdDev\_Fp – image standard deviation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageStdDev_Fp(mllib_d64 *sdev, const mllib_image *img,
 const mllib_d64 *mean);
```

**Description** The `mllib_ImageStdDev_Fp()` function computes the standard deviation for each channel in the floating-point source image.

It uses the following equation:

$$sdev[i] = \left\{ \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (img[x][y][i] - mean[i])^2 \right\}^{0.5}$$

where, in the case of `mean == NULL`,

$$mean[i] = \frac{1}{w \cdot h} * \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} img[x][y][i]$$

**Parameters** The function takes the following arguments:

- sdev* Pointer to standard deviation array, whose size is the number of channels in the source image. `sdev[i]` contains the standard deviation of channel `i`.
- img* Pointer to input image.
- mean* Pointer to pre-computed mean array for each channel. (If `NULL`, it will be computed.) `mean[i]` contains the mean of channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageMean\(3MLIB\)](#), [mllib\\_ImageMean\\_Fp\(3MLIB\)](#), [mllib\\_ImageStdDev\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub1\_Fp\_Inp – subtraction, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSub1_Fp_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageSub1_Fp_Inp()` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] - \text{src2}[x][y][i]$$

**Parameters** The function takes the following arguments:

- src1dst*     Pointer to first source and destination image.
- src2*        Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSub\(3MLIB\)](#), [mllib\\_ImageSub\\_Fp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub1\_Inp – subtraction, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSub1_Inp(mllib_image *src1dst,
 const mllib_image *src2);
```

**Description** The `mllib_ImageSub1_Inp()` function subtracts the second source image from the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$src1dst[x][y][i] = src1dst[x][y][i] - src2[x][y][i]$$

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_ImageSub\(3MLIB\)](#), [mllib\\_ImageSub\\_Fp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub2\_Fp\_Inp – subtraction, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSub2_Fp_Inp(mllib_image *src2dst,
 const mllib_image *src1);
```

**Description** The `mllib_ImageSub2_Fp_Inp()` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src2dst}[x][y][i] = \text{src1}[x][y][i] - \text{src2dst}[x][y][i]$$

**Parameters** The function takes the following arguments:

- src2dst*     Pointer to second source and destination image.
- src1*        Pointer to first source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSub\(3MLIB\)](#), [mllib\\_ImageSub\\_Fp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub2\_Inp – subtraction, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageSub2_Inp(mllib_image *src2dst,
 const mllib_image *src1);
```

**Description** The `mllib_ImageSub2_Inp()` function subtracts the second source image from the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$\text{src2dst}[x][y][i] = \text{src1}[x][y][i] - \text{src2dst}[x][y][i]$$

**Parameters** The function takes the following arguments:

*src2dst*      Pointer to second source and destination image.

*src1*          Pointer to first source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSub\(3MLIB\)](#), [mllib\\_ImageSub\\_Fp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub – subtraction

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageSub(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageSub()` function subtracts the second source image from the first source image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] - src2[x][y][i]$$

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination image.
- src1*     Pointer to first source image.
- src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSub\\_Fp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageSub1\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageSub2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSub\_Fp – subtraction

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageSub_Fp(mllib_image *dst, const mllib_image *src1,
 const mllib_image *src2);
```

**Description** The `mllib_ImageSub_Fp()` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis.

It uses the following equation:

$$dst[x][y][i] = src1[x][y][i] - src2[x][y][i]$$

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSub\(3MLIB\)](#), [mllib\\_ImageSub1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub1\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageSub2\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSubsampleAverage, mllib\_ImageSubsampleAverage\_Fp – subsamples an image with a box filter

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSubsampleAverage(mllib_image *dst,
 const mllib_image *src, mllib_d64 xscale, mllib_d64 yscale);

mllib_status mllib_ImageSubsampleAverage_Fp(mllib_image *dst,
 const mllib_image *src, mllib_d64 xscale, mllib_d64 yscale);
```

**Description** Each function scales an image down with an adaptive box filter.

The subsampling algorithm performs the scaling operation by averaging all the pixel values from a block in the source image that correspond to the destination pixel.

The width and height of the source block for a destination pixel are computed as:

```
blockX = (int)ceil(1.0/xscale);
blockY = (int)ceil(1.0/yscale);
```

If we denote a pixel's location in an image by its column number and row number (both counted from 0), the destination pixel at (i, j) is backward mapped to the source block whose upper-left corner pixel is at (xValues[i], yValues[j]), where

```
xValues[i] = (int)(i/xscale + 0.5);
yValues[j] = (int)(j/yscale + 0.5);
```

The width and height of the filled area in the destination are restricted by

```
dstW = (int)(srcWidth * xscale);
dstH = (int)(srcHeight * yscale);
```

where srcWidth and srcHeight are width and height of the source image.

Since the block size in source is defined from scale factors with roundup, some blocks (the rightmost and the bottommost blocks) may overrun the border of the source image by 1 pixel. In this case, such blocks are moved by 1 pixel to left/up direction in order to be inside of the source image.

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*xscale*     X scale factor.  $0.0 < xscale \leq 1.0$ .

*yscale*     Y scale factor.  $0.0 < yscale \leq 1.0$ .



**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageSubsampleBinaryToGray\(3MLIB\)](#), [mllib\\_ImageFilteredSubsample\(3MLIB\)](#), [mllib\\_ImageZoomTranslate\(3MLIB\)](#), [mllib\\_ImageZoomTranslate\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageSubsampleBinaryToGray – subsamples a binary image and converts it to a grayscale image

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageSubsampleBinaryToGray(mllib_image *dst,
 const mllib_image *src, mllib_d64 xscale, mllib_d64 yscale,
 const mllib_u8 *lutGray);
```

**Description** The `mllib_ImageSubsampleBinaryToGray()` function subsamples a binary (MLIB\_BIT) image and converts it to a grayscale (MLIB\_BYTE) image.

The subsampling algorithm performs the scaling operation by accumulating all the bits in the source image that correspond to the destination pixel and, based on the x and y scaling factors, reserving consecutive indexes in the colormap for the maximum number of gray levels possible in the destination image. The destination image pixel values of this function are either gray levels or indexes (if `lutGray==NULL`).

For representing the source block of pixels that is used to determine destination pixel values, the index 0 represents a block with no 1's (all 0's), the index 1 represents a block with a single 1, and so on. If the scaling factors require a fractional block of source pixels to determine a destination pixel value, the block size is rounded up. For example, if a 2.2-by-2.2 block of source pixels would be required to determine destination pixel values, a 3-by-3 block is used, resulting in 10 possible gray levels and therefore 10 colormap indexes, whose values are 0 through 9.

The width and height of the source block for a destination pixel are computed as:

```
blockX = (int)ceil(1.0/xscale);
blockY = (int)ceil(1.0/yscale);
```

If we denote a pixel's location in an image by its column number and row number (both counted from 0), the destination pixel at  $(i, j)$  is backward mapped to the source block whose upper-left corner pixel is at  $(xValues[i], yValues[j])$ , where

```
xValues[i] = (int)(i/xscale + 0.5);
yValues[j] = (int)(j/yscale + 0.5);
```

The width and height of the filled area in the destination are restricted by

```
dstW = (int)(srcWidth * xscale);
dstH = (int)(srcHeight * yscale);
```

where `srcWidth` and `srcHeight` are width and height of the source image.

Since the block size in source is defined from scale factors with roundup, some blocks (the rightmost and the bottommost blocks) may overrun the border of the source image by 1 pixel. In this case, such blocks are moved by 1 pixel to left/up direction in order to be inside of the source image.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image . It must be of type MLIB\_BYTE and have just one channel.
- src* Pointer to source image. It must be of type MLIB\_BIT and have just one channel.
- xscale* X scale factor.  $0.0 < xscale \leq 1.0$ .
- yscale* Y scale factor.  $0.0 < yscale \leq 1.0$ .
- lutGray* Pointer to a grayscale lookup-table.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoomTranslateToGray\(3MLIB\)](#), [mllib\\_ImageSubsampleAverage\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageTestFlags – test flags

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`int mllib_ImageTestFlags(const mllib_image *img, mllib_s32 flags);`

**Description** The `mllib_ImageTestFlags()` function tests the flags for a combination of the following predefined characteristics. Note that the result of zero means the conditions are satisfied.

```
MLIB_IMAGE_ALIGNED64 /* data address is 64-byte aligned */
MLIB_IMAGE_ALIGNED8 /* data address is 8-byte aligned */
MLIB_IMAGE_ALIGNED4 /* data address is 4-byte aligned */
MLIB_IMAGE_ALIGNED2 /* data address is 2-byte aligned */
MLIB_IMAGE_WIDTH8X /* width is multiple of 8 */
MLIB_IMAGE_WIDTH4X /* width is multiple of 4 */
MLIB_IMAGE_WIDTH2X /* width is multiple of 2 */
MLIB_IMAGE_HEIGHT8X /* height is multiple of 8 */
MLIB_IMAGE_HEIGHT4X /* height is multiple of 4 */
MLIB_IMAGE_HEIGHT2X /* height is multiple of 2 */
MLIB_IMAGE_STRIDE8X /* stride is multiple of 8 */
MLIB_IMAGE_ONEVECTOR /* stride is equal to width in bytes */
MLIB_IMAGE_USERALLOCATED /* data space has been allocated by user */
MLIB_IMAGE_ATTRIBUTESET /* image attribute flags have been set */
```

**Parameters** The function takes the following arguments:

- img*      Pointer to a mediaLib image structure.
- flags*     Combination of a set of characteristics to be tested. It is formed by logically Oring one or more individual predefined characteristics.

**Return Values** The function returns an integer value containing results of test. Condition = 0 if satisfied; otherwise, Condition != 0.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageGetFlags\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh1 – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh1(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *thresh, const mllib_s32 *ghigh, const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh1()` function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

The data type of the destination image can be `MLIB_BIT` or can be the same as the data type of the source image.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*thresh*    Threshold value. `thresh[i]` contains the threshold for channel *i*.

*ghigh*     High output level. `ghigh[i]` contains the high output level for channel *i*.

*glow*      Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh4_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```

**Name** mllib\_ImageThresh1\_Fp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageThresh1_Fp(mllib_image *dst, const mllib_image *src,
 const mllib_d64 *thresh, const mllib_d64 *ghigh, const mllib_d64 *glow);
```

**Description** The `mllib_ImageThresh1_Fp()` function compares each pixel in the floating-point source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

The data type of the destination image can be `MLIB_BIT` or can be the same as the data type of the source image.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

*dst*            Pointer to destination image.

*src*            Pointer to source image.

*thresh*        Threshold value. `thresh[i]` contains the threshold for channel *i*.

*ghigh*        High output level. `ghigh[i]` contains the high output level for channel *i*.

*glow*        Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh4_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```



**Name** mlib\_ImageThresh1\_Fp\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_ImageThresh1_Fp_Inp(mlib_image *srcdst,
 const mlib_d64 *thresh, const mlib_d64 *ghigh, const mlib_d64 *glow);
```

**Description** The `mlib_ImageThresh1_Fp_Inp()` function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

```
srcdst[x][y][i] = glow[i] if srcdst[x][y][i] ≤ thresh[i]
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to source and destination image.

*thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.

*ghigh*     High output level. `ghigh[i]` contains the high output level for channel *i*.

*glow*     Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageThresh1\(3MLIB\)](#), [mlib\\_ImageThresh1\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh1\\_Inp\(3MLIB\)](#), [mlib\\_ImageThresh2\(3MLIB\)](#), [mlib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mlib\\_ImageThresh3\(3MLIB\)](#), [mlib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mlib\\_ImageThresh4\(3MLIB\)](#), [mlib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mlib\\_ImageThresh5\(3MLIB\)](#), [mlib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mlib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh1\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh1_Inp(mllib_image *srcdst,
 const mllib_s32 *thresh, const mllib_s32 *ghigh, const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh1_Inp()` function compares each pixel in the image to a threshold value on a per-channel basis. If the pixel is less than or equal to the threshold value, then it is reset to the low output level. If the pixel is greater than the threshold value, then it is reset to the high output level.

It uses the following equation:

```
srcdst[x][y][i] = glow[i] if srcdst[x][y][i] ≤ thresh[i]
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thresh*    Threshold value. `thresh[i]` contains the threshold for channel *i*.
- ghigh*     High output level. `ghigh[i]` contains the high output level for channel *i*.
- glow*      Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh2 – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh2(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *thresh, const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh2()` function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = src[x][y][i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.

*glow*       Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh2\_Fp – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh2_Fp(mllib_image *dst, const mllib_image *src,
 const mllib_d64 *thresh, const mllib_d64 *glow);
```

**Description** The `mllib_ImageThresh2_Fp()` function compares each pixel in the floating-point source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = src[x][y][i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.
- glow*       Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageThresh2\_Fp\_Inp – image thresholding

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_ImageThresh2_Fp_Inp(mlib_image *srcdst,
 const mlib_d64 *thresh, const mlib_d64 *glow);
```

**Description** The `mlib_ImageThresh2_Fp_Inp()` function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

$$\text{srcdst}[x][y][i] = \text{glow}[i] \quad \text{if } \text{srcdst}[x][y][i] \leq \text{thresh}[i]$$

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to source and destination image.

*thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.

*glow*       Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageThresh1(3MLIB)`, `mlib_ImageThresh1_Fp(3MLIB)`,  
`mlib_ImageThresh1_Fp_Inp(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`,  
`mlib_ImageThresh2(3MLIB)`, `mlib_ImageThresh2_Fp(3MLIB)`,  
`mlib_ImageThresh2_Inp(3MLIB)`, `mlib_ImageThresh3(3MLIB)`,  
`mlib_ImageThresh3_Fp(3MLIB)`, `mlib_ImageThresh3_Fp_Inp(3MLIB)`,  
`mlib_ImageThresh3_Inp(3MLIB)`, `mlib_ImageThresh4(3MLIB)`,  
`mlib_ImageThresh4_Fp(3MLIB)`, `mlib_ImageThresh4_Fp_Inp(3MLIB)`,  
`mlib_ImageThresh4_Inp(3MLIB)`, `mlib_ImageThresh5(3MLIB)`,  
`mlib_ImageThresh5_Fp(3MLIB)`, `mlib_ImageThresh5_Fp_Inp(3MLIB)`,  
`mlib_ImageThresh5_Inp(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageThresh2\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh2_Inp(mllib_image *srcdst,
 const mllib_s32 *thresh, const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh2_Inp()` function compares each pixel in the source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

$$srcdst[x][y][i] = glow[i] \text{ if } srcdst[x][y][i] \leq thresh[i]$$

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.
- glow*       Low output level. `glow[i]` contains the low output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageThresh1(3MLIB)`, `mllib_ImageThresh1_Fp(3MLIB)`,  
`mllib_ImageThresh1_Fp_Inp(3MLIB)`, `mllib_ImageThresh1_Inp(3MLIB)`,  
`mllib_ImageThresh2(3MLIB)`, `mllib_ImageThresh2_Fp(3MLIB)`,  
`mllib_ImageThresh2_Fp_Inp(3MLIB)`, `mllib_ImageThresh3(3MLIB)`,  
`mllib_ImageThresh3_Fp(3MLIB)`, `mllib_ImageThresh3_Fp_Inp(3MLIB)`,  
`mllib_ImageThresh3_Inp(3MLIB)`, `mllib_ImageThresh4(3MLIB)`,  
`mllib_ImageThresh4_Fp(3MLIB)`, `mllib_ImageThresh4_Fp_Inp(3MLIB)`,  
`mllib_ImageThresh4_Inp(3MLIB)`, `mllib_ImageThresh5(3MLIB)`,  
`mllib_ImageThresh5_Fp(3MLIB)`, `mllib_ImageThresh5_Fp_Inp(3MLIB)`,  
`mllib_ImageThresh5_Inp(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageThresh3 – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh3(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *thresh, const mllib_s32 *ghigh);
```

**Description** The `mllib_ImageThresh3()` function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

*dst*            Pointer to destination image.

*src*            Pointer to source image.

*thresh*        Threshold value. `thresh[i]` contains the threshold for channel *i*.

*ghigh*         High output level. `ghigh[i]` contains the high output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh3\_Fp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh3_Fp(mllib_image *dst, const mllib_image *src,
 const mllib_d64 *thresh, const mllib_d64 *ghigh);
```

**Description** The `mllib_ImageThresh3_Fp()` function compares each pixel in the floating-point source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thresh[i]
```

**Parameters** The function takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to source image.
- thresh*        Threshold value. `thresh[i]` contains the threshold for channel *i*.
- ghigh*         High output level. `ghigh[i]` contains the high output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageThresh3\_Fp\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh3_Fp_Inp(mllib_image *srcdst,
 const mllib_d64 *thresh, const mllib_d64 *ghigh);
```

**Description** The `mllib_ImageThresh3_Fp_Inp()` function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

$$srcdst[x][y][i] = ghigh[i] \text{ if } srcdst[x][y][i] > thresh[i]$$

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.
- ghigh*     High output level. `ghigh[i]` contains the high output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageThresh1(3MLIB)`, `mllib_ImageThresh1_Fp(3MLIB)`, `mllib_ImageThresh1_Fp_Inp(3MLIB)`, `mllib_ImageThresh1_Inp(3MLIB)`, `mllib_ImageThresh2(3MLIB)`, `mllib_ImageThresh2_Fp(3MLIB)`, `mllib_ImageThresh2_Fp_Inp(3MLIB)`, `mllib_ImageThresh2_Inp(3MLIB)`, `mllib_ImageThresh3(3MLIB)`, `mllib_ImageThresh3_Fp(3MLIB)`, `mllib_ImageThresh3_Inp(3MLIB)`, `mllib_ImageThresh4(3MLIB)`, `mllib_ImageThresh4_Fp(3MLIB)`, `mllib_ImageThresh4_Fp_Inp(3MLIB)`, `mllib_ImageThresh4_Inp(3MLIB)`, `mllib_ImageThresh5(3MLIB)`, `mllib_ImageThresh5_Fp(3MLIB)`, `mllib_ImageThresh5_Fp_Inp(3MLIB)`, `mllib_ImageThresh5_Inp(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageThresh3\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh3_Inp(mllib_image *srcdst,
 const mlib_s32 *thresh, const mlib_s32 *ghigh);
```

**Description** The `mllib_ImageThresh3_Inp()` function compares each pixel in the source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

`srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thresh[i]`

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thresh*     Threshold value. `thresh[i]` contains the threshold for channel *i*.
- ghigh*     High output level. `ghigh[i]` contains the high output level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh5\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageThresh4 – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh4(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *thigh, const mllib_s32 *tlow, const mllib_s32 *ghigh,
 const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh4()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is less than the lower threshold value, *tlow*, then the destination pixel is set to the lower output level, *glow*. If the pixel is greater than the higher threshold value, *thigh*, then the destination pixel is set to the higher output level, *ghigh*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] < tlow[i]
dst[x][y][i] = src[x][y][i] if tlow[i] ≤ src[x][y][i] ≤ thigh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*thigh*      High threshold value. `thigh[i]` holds the high threshold for channel *i*.

*tlow*       Low threshold value. `tlow[i]` holds the low threshold for channel *i*.

*ghigh*      High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel *i*.

*glow*       Low output grayscale level. `glow[i]` holds the low output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh3(3MLIB),mllib_ImageThresh3_Fp(3MLIB),
mllib_ImageThresh3_Fp_Inp(3MLIB),mllib_ImageThresh3_Inp(3MLIB),
mllib_ImageThresh4_Fp(3MLIB),mllib_ImageThresh4_Fp_Inp(3MLIB),
mllib_ImageThresh4_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```

**Name** mllib\_ImageThresh4\_Fp – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh4_Fp(mllib_image *dst, const mllib_image *src,
 const mllib_d64 *thigh, const mllib_d64 *tlow, const mllib_d64 *ghigh,
 const mllib_d64 *glow);
```

**Description** The `mllib_ImageThresh4_Fp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is less than the lower threshold value, *tlow*, then the destination pixel is set to the lower output level, *glow*. If the pixel is greater than the higher threshold value, *thigh*, then the destination pixel is set to the higher output level, *ghigh*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = glow[i] if src[x][y][i] < tlow[i]
dst[x][y][i] = src[x][y][i] if tlow[i] ≤ src[x][y][i] ≤ thigh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to source image.

*thigh*    High threshold value. `thigh[i]` holds the high threshold for channel *i*.

*tlow*     Low threshold value. `tlow[i]` holds the low threshold for channel *i*.

*ghigh*    High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel *i*.

*glow*     Low output grayscale level. `glow[i]` holds the low output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh3(3MLIB),mllib_ImageThresh3_Fp(3MLIB),
mllib_ImageThresh3_Fp_Inp(3MLIB),mllib_ImageThresh3_Inp(3MLIB),
mllib_ImageThresh4(3MLIB),mllib_ImageThresh4_Fp_Inp(3MLIB),
mllib_ImageThresh4_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```

**Name** mllib\_ImageThresh4\_Fp\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh4_Fp_Inp(mllib_image *srcdst,
 const mllib_d64 *thigh, const mllib_d64 *tlow, const mllib_d64 *ghigh,
 const mllib_d64 *glow);
```

**Description** The `mllib_ImageThresh4_Fp_Inp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is less than the lower threshold value, *tlow*, then the destination pixel is set to the lower output level, *glow*. If the pixel is greater than the higher threshold value, *thigh*, then the destination pixel is set to the higher output level, *ghigh*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
srcdst[x][y][i] = glow[i] if srcdst[x][y][i] < tlow[i]
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thigh*     High threshold value. `thigh[i]` holds the high threshold for channel *i*.
- tlow*      Low threshold value. `tlow[i]` holds the low threshold for channel *i*.
- ghigh*     High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel *i*.
- glow*      Low output grayscale level. `glow[i]` holds the low output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh4(3MLIB),mllib_ImageThresh4_Fp(3MLIB),
mllib_ImageThresh4_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```



**Name** mllib\_ImageThresh4\_Inp – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageThresh4_Inp(mllib_image *srcdst,
 const mllib_s32 *thigh, const mllib_s32 *tlow, const mllib_s32 *ghigh,
 const mllib_s32 *glow);
```

**Description** The `mllib_ImageThresh4_Inp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is less than the lower threshold value, *tlow*, then the destination pixel is set to the lower output level, *glow*. If the pixel is greater than the higher threshold value, *thigh*, then the destination pixel is set to the higher output level, *ghigh*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
srcdst[x][y][i] = glow[i] if srcdst[x][y][i] < tlow[i]
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to source and destination image.

*thigh*     High threshold value. `thigh[i]` holds the high threshold for channel *i*.

*tlow*       Low threshold value. `tlow[i]` holds the low threshold for channel *i*.

*ghigh*     High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel *i*.

*glow*       Low output grayscale level. `glow[i]` holds the low output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#),  
[mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#),

```
mllib_ImageThresh4(3MLIB),mllib_ImageThresh4_Fp(3MLIB),
mllib_ImageThresh4_Fp_Inp(3MLIB),mllib_ImageThresh5(3MLIB),
mllib_ImageThresh5_Fp(3MLIB),mllib_ImageThresh5_Fp_Inp(3MLIB),
mllib_ImageThresh5_Inp(3MLIB),attributes(5)
```

**Name** mllib\_ImageThresh5 – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageThresh5(mllib_image *dst, const mllib_image *src,
 const mllib_s32 *thigh, const mllib_s32 *tlow, const mllib_s32 *gray);
```

**Description** The `mllib_ImageThresh5()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is in between the lower threshold value, *tlow*, and the higher threshold value, *thigh*, (inclusive on both sides), then the destination pixel is set to the value *gray*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i] if src[x][y][i] < tlow[i]
dst[x][y][i] = gray[i] if tlow[i] ≤ src[x][y][i] ≤ thigh[i]
dst[x][y][i] = src[x][y][i] if src[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*thigh*      High threshold value. `thigh[i]` holds the high threshold for channel *i*.

*tlow*       Low threshold value. `tlow[i]` holds the low threshold for channel *i*.

*gray*       Output grayscale level. `gray[i]` holds the output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageThresh1(3MLIB)`, `mllib_ImageThresh1_Fp(3MLIB)`,  
`mllib_ImageThresh1_Fp_Inp(3MLIB)`, `mllib_ImageThresh1_Inp(3MLIB)`,  
`mllib_ImageThresh2(3MLIB)`, `mllib_ImageThresh2_Fp(3MLIB)`,  
`mllib_ImageThresh2_Fp_Inp(3MLIB)`, `mllib_ImageThresh2_Inp(3MLIB)`,  
`mllib_ImageThresh3(3MLIB)`, `mllib_ImageThresh3_Fp(3MLIB)`,  
`mllib_ImageThresh3_Fp_Inp(3MLIB)`, `mllib_ImageThresh3_Inp(3MLIB)`,  
`mllib_ImageThresh4(3MLIB)`, `mllib_ImageThresh4_Fp(3MLIB)`,  
`mllib_ImageThresh4_Fp_Inp(3MLIB)`, `mllib_ImageThresh4_Inp(3MLIB)`,  
`mllib_ImageThresh5_Fp(3MLIB)`, `mllib_ImageThresh5_Fp_Inp(3MLIB)`,  
`mllib_ImageThresh5_Inp(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageThresh5\_Fp – image thresholding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageThresh5_Fp(mllib_image *dst, const mllib_image *src,
 const mlib_d64 *thigh, const mlib_d64 *tlow, const mlib_d64 *gray);
```

**Description** The `mllib_ImageThresh5_Fp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is in between the lower threshold value, *tlow*, and the higher threshold value, *thigh*, (inclusive on both sides), then the destination pixel is set to the value *gray*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

```
dst[x][y][i] = src[x][y][i] if src[x][y][i] < tlow[i]
dst[x][y][i] = gray[i] if tlow[i] ≤ src[x][y][i] ≤ thigh[i]
dst[x][y][i] = src[x][y][i] if src[x][y][i] > thigh[i]
```

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- thigh*      High threshold value. `thigh[i]` holds the high threshold for channel *i*.
- tlow*       Low threshold value. `tlow[i]` holds the low threshold for channel *i*.
- gray*       Output grayscale level. `gray[i]` holds the output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageThresh1(3MLIB)`, `mllib_ImageThresh1_Fp(3MLIB)`,  
`mllib_ImageThresh1_Fp_Inp(3MLIB)`, `mllib_ImageThresh1_Inp(3MLIB)`,  
`mllib_ImageThresh2(3MLIB)`, `mllib_ImageThresh2_Fp(3MLIB)`,  
`mllib_ImageThresh2_Fp_Inp(3MLIB)`, `mllib_ImageThresh2_Inp(3MLIB)`,  
`mllib_ImageThresh3(3MLIB)`, `mllib_ImageThresh3_Fp(3MLIB)`,  
`mllib_ImageThresh3_Fp_Inp(3MLIB)`, `mllib_ImageThresh3_Inp(3MLIB)`,  
`mllib_ImageThresh4(3MLIB)`, `mllib_ImageThresh4_Fp(3MLIB)`,  
`mllib_ImageThresh4_Fp_Inp(3MLIB)`, `mllib_ImageThresh4_Inp(3MLIB)`,  
`mllib_ImageThresh5(3MLIB)`, `mllib_ImageThresh5_Fp_Inp(3MLIB)`,  
`mllib_ImageThresh5_Inp(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageThresh5\_Fp\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh5_Fp_Inp(mllib_image *srcdst,
 const mllib_d64 *thigh, const mllib_d64 *tlow, const mllib_d64 *gray);
```

**Description** The `mllib_ImageThresh5_Fp_Inp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is in between the lower threshold value, *tlow*, and the higher threshold value, *thigh*, (inclusive on both sides), then the destination pixel is set to the value *gray*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

$$srcdst[x][y][i] = gray[i] \text{ if } tlow[i] \leq srcdst[x][y][i] \leq thigh[i]$$

**Parameters** The function takes the following arguments:

*srcdst*     Pointer to source and destination image.

*thigh*     High threshold value. `thigh[i]` holds the high threshold for channel *i*.

*tlow*     Low threshold value. `tlow[i]` holds the low threshold for channel *i*.

*gray*     Output grayscale level. `gray[i]` holds the output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageThresh1(3MLIB)`, `mllib_ImageThresh1_Fp(3MLIB)`,  
`mllib_ImageThresh1_Fp_Inp(3MLIB)`, `mllib_ImageThresh1_Inp(3MLIB)`,  
`mllib_ImageThresh2(3MLIB)`, `mllib_ImageThresh2_Fp(3MLIB)`,  
`mllib_ImageThresh2_Fp_Inp(3MLIB)`, `mllib_ImageThresh2_Inp(3MLIB)`,  
`mllib_ImageThresh3(3MLIB)`, `mllib_ImageThresh3_Fp(3MLIB)`,  
`mllib_ImageThresh3_Fp_Inp(3MLIB)`, `mllib_ImageThresh3_Inp(3MLIB)`,  
`mllib_ImageThresh4(3MLIB)`, `mllib_ImageThresh4_Fp(3MLIB)`,  
`mllib_ImageThresh4_Fp_Inp(3MLIB)`, `mllib_ImageThresh4_Inp(3MLIB)`,  
`mllib_ImageThresh5(3MLIB)`, `mllib_ImageThresh5_Fp(3MLIB)`,  
`mllib_ImageThresh5_Inp(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageThresh5\_Inp – image thresholding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_ImageThresh5_Inp(mllib_image *srcdst,
 const mllib_s32 *thigh, const mllib_s32 *tlow, const mllib_s32 *gray);
```

**Description** The `mllib_ImageThresh5_Inp()` function compares each pixel in the source image to two threshold values, *tlow* and *thigh*. If the pixel is in between the lower threshold value, *tlow*, and the higher threshold value, *thigh*, (inclusive on both sides), then the destination pixel is set to the value *gray*. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

$$srcdst[x][y][i] = gray[i] \text{ if } tlow[i] \leq srcdst[x][y][i] \leq thigh[i]$$

**Parameters** The function takes the following arguments:

- srcdst*     Pointer to source and destination image.
- thigh*     High threshold value. `thigh[i]` holds the high threshold for channel *i*.
- tlow*     Low threshold value. `tlow[i]` holds the low threshold for channel *i*.
- gray*     Output grayscale level. `gray[i]` holds the output grayscale level for channel *i*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageThresh1\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh1\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh2\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh3\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Fp\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh4\\_Inp\(3MLIB\)](#), [mllib\\_ImageThresh5\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\(3MLIB\)](#), [mllib\\_ImageThresh5\\_Fp\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_ImageXor – Xor

**Synopsis**

```
cc [flag...] file... -llib [library...]
#include <mlib.h>
```

```
mlib_status mlib_ImageXor(mlib_image *dst, const mlib_image *src1,
 const mlib_image *src2);
```

**Description** The `mlib_ImageXor()` function computes the exclusive Or of the first source image with the second source image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = \text{src1}[x][y][i] \wedge \text{src2}[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src1*     Pointer to first source image.

*src2*     Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_ImageXor\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageXor\_Inp – Xor, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

`mllib_status mllib_ImageXor_Inp(mllib_image *src1dst, const mllib_image *src2);`

**Description** The `mllib_ImageXor_Inp()` function computes the exclusive Or of the first source image with the second source image on a pixel-by-pixel basis, in place.

It uses the following equation:

$$src1dst[x][y][i] = src1dst[x][y][i] \wedge src2[x][y][i]$$

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters** The function takes the following arguments:

*src1dst*      Pointer to first source and destination image.

*src2*          Pointer to second source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageXor\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageXProj – image X projection

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageXProj(mllib_d64 *xproj, const mllib_image *img);
```

**Description** The `mllib_ImageXProj()` function computes the sum of the pixels in each column of the source image.

The image must be a single-channel image.

It uses the following equation:

$$xproj[x] = \sum_{y=0}^{h-1} img[x][y][0]$$

where  $x = 0, 1, \dots, w - 1$ .

**Parameters** The function takes the following arguments:

*xproj*     Pointer to X-projection vector, where length is equal to the number of columns in the source image (in other words, the image width).

*img*        Pointer to an input image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageXProj\\_Fp\(3MLIB\)](#), [mllib\\_ImageYProj\(3MLIB\)](#), [mllib\\_ImageYProj\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageXProj\_Fp – image X projection

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageXProj_Fp(mllib_d64 *xproj, const mllib_image *img);`

**Description** The `mllib_ImageXProj_Fp()` function computes the sum of the pixels in each column of the floating-point source image.

The image must be a single-channel image.

It uses the following equation:

$$xproj[x] = \sum_{y=0}^{h-1} img[x][y][0]$$

where  $x = 0, 1, \dots, w - 1$ .

**Parameters** The function takes the following arguments:

*xproj*     Pointer to X-projection vector, where length is equal to the number of columns in the source image (in other words, the image width).

*img*       Pointer to an input image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageXProj\(3MLIB\)](#), [mllib\\_ImageYProj\(3MLIB\)](#), [mllib\\_ImageYProj\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageYProj – image Y projection

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageYProj(mllib_d64 *yproj, const mllib_image *img);
```

**Description** The `mllib_ImageYProj()` function computes the sum of the pixels in each row of the source image.

The image must be a single-channel image.

It uses the following equation:

$$yproj[y] = \sum_{x=0}^{w-1} img[x][y][0]$$

where  $y = 0, 1, \dots, h - 1$ .

**Parameters** The function takes the following arguments:

*yproj*     Pointer to Y-projection vector, where length is equal to the number of rows in the source image (in other words, the image height).

*img*        Pointer to an input image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageXProj\(3MLIB\)](#), [mllib\\_ImageXProj\\_Fp\(3MLIB\)](#), [mllib\\_ImageYProj\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageYProj\_Fp – image Y projection

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_ImageYProj_Fp(mllib_d64 *yproj, const mllib_image *img);`

**Description** The `mllib_ImageYProj_Fp()` function computes the sum of the pixels in each row of the floating-point source image.

The image must be a single-channel image.

It uses the following equation:

$$yproj[y] = \sum_{x=0}^{w-1} img[x][y][0]$$

where  $y = 0, 1, \dots, h - 1$ .

**Parameters** The function takes the following arguments:

*yproj*     Pointer to Y-projection vector, where length is equal to the number of rows in the source image (in other words, the image height).

*img*       Pointer to an input image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageXProj\(3MLIB\)](#), [mllib\\_ImageXProj\\_Fp\(3MLIB\)](#), [mllib\\_ImageYProj\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoom – zoom

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageZoom(mllib_image *dst, const mllib_image *src,
 mllib_d64 zoomx, mllib_d64 zoomy, mllib_filter filter, mllib_edge edge);
```

**Description** The `mllib_ImageZoom()` function will enlarge or minify the source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*zoomx* X zoom factor.  $zoomx > 0.0$ .

*zoomy* Y zoom factor.  $zoomy > 0.0$ .

*filter* Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge* Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_EXTEND_INDEF
MLIB_EDGE_SRC_PADDED
```

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoom\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomIn2X\(3MLIB\)](#),  
[mllib\\_ImageZoomIn2X\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomIn2XIndex\(3MLIB\)](#),  
[mllib\\_ImageZoomIndex\(3MLIB\)](#), [mllib\\_ImageZoomOut2X\(3MLIB\)](#),  
[mllib\\_ImageZoomOut2X\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomOut2XIndex\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomBlend – image scaling with alpha blending

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomBlend(mllib_image *dst, const mllib_image *src,
 mllib_d64 zoomx, mllib_d64 zoomy, mllib_filter filter, mllib_edge edge,
 mllib_blend blend, mllib_s32 alpha, mllib_s32 cmask);
```

**Description** The `mllib_ImageZoomBlend()` function will enlarge or minify the source image by the X and Y zoom factors and blend it with the destination image.

This function is a special case of `mllib_ImageZoomTranslateBlend()` with the center of the source image being mapped to the center of the destination image.

The center of the upper-left corner pixel of an image is considered to be located at  $(0.5, 0.5)$ .

Both *src* and *dst* must be of type `MLIB_BYTE`. They can have either 3 or 4 channels.

The *src* image cannot have width or height larger than 32767.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to first source image.

*zoomx* X zoom factor.  $\text{zoomx} > 0.0$ .

*zoomy* Y zoom factor.  $\text{zoomy} > 0.0$ .

*filter* Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge* Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_EXTEND_INDEF
MLIB_EDGE_SRC_PADDED
```

*blend* Type of alpha blending. It can be one of the following:

```
MLIB_BLEND_GTK_SRC
MLIB_BLEND_GTK_SRC_OVER
MLIB_BLEND_GTK_SRC_OVER2
```

*alpha* Overall alpha for blending.

*cmask*    Channel mask to indicate the alpha channel.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_ImageZoomTranslateBlend\(3MLIB\)](#), [mllib\\_ImageZoomTranslateTableBlend\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageZoom\_Fp – zoom

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoom_Fp(mllib_image *dst, const mllib_image *src,
 mlib_d64 zoomx, mlib_d64 zoomy, mlib_filter filter, mlib_edge edge);
```

**Description** The `mllib_ImageZoom_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*zoomx*     X zoom factor. `zoomx > 0.0`.

*zoomy*     Y zoom factor. `zoomy > 0.0`.

*filter*     Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge*       Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_EXTEND_INDEF
MLIB_EDGE_SRC_PADDED
```

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_ImageZoom(3MLIB)`, `mllib_ImageZoomIn2X(3MLIB)`, `mllib_ImageZoomIn2X_Fp(3MLIB)`,  
`mllib_ImageZoomIn2XIndex(3MLIB)`, `mllib_ImageZoomIndex(3MLIB)`,  
`mllib_ImageZoomOut2X(3MLIB)`, `mllib_ImageZoomOut2X_Fp(3MLIB)`,  
`mllib_ImageZoomOut2XIndex(3MLIB)`, `attributes(5)`

**Name** mlib\_ImageZoomIn2X – 2X zoom

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_ImageZoomIn2X(mlib_image *dst, const mlib_image *src,
 mlib_filter filter, mlib_edge edge);
```

**Description** The `mlib_ImageZoomIn2X()` function enlarges the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*filter* Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge* Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageZoom(3MLIB)`, `mlib_ImageZoom_Fp(3MLIB)`, `mlib_ImageZoomIn2X_Fp(3MLIB)`, `mlib_ImageZoomIn2XIndex(3MLIB)`, `mlib_ImageZoomIndex(3MLIB)`, `mlib_ImageZoomOut2X(3MLIB)`, `mlib_ImageZoomOut2X_Fp(3MLIB)`, `mlib_ImageZoomOut2XIndex(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageZoomIn2X\_Fp – 2X zoom

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomIn2X_Fp(mllib_image *dst, const mllib_image *src,
 mllib_filter filter, mllib_edge edge);
```

**Description** The `mllib_ImageZoomIn2X_Fp()` function enlarges the floating-point source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination image.
- src*        Pointer to source image.
- filter*     Type of resampling filter. It can be one of the following:
- MLIB\_NEAREST
  - MLIB\_BILINEAR
  - MLIB\_BICUBIC
  - MLIB\_BICUBIC2
- edge*       Type of edge condition. It can be one of the following:
- MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_OP\_NEAREST
  - MLIB\_EDGE\_SRC\_EXTEND
  - MLIB\_EDGE\_SRC\_PADDED

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageZoom(3MLIB)`, `mllib_ImageZoom_Fp(3MLIB)`, `mllib_ImageZoomIn2X(3MLIB)`,  
`mllib_ImageZoomIn2XIndex(3MLIB)`, `mllib_ImageZoomIndex(3MLIB)`,  
`mllib_ImageZoomOut2X(3MLIB)`, `mllib_ImageZoomOut2X_Fp(3MLIB)`,  
`mllib_ImageZoomOut2XIndex(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageZoomIn2XIndex – 2X zoom on color-indexed image

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomIn2XIndex(mllib_image *dst,
 const mllib_image *src, mllib_filter filter, mllib_edge edge,
 const void *colormap);
```

**Description** The `mllib_ImageZoomIn2XIndex()` function enlarges the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*filter* Type of resampling filter. It can be one of the following:

`MLIB_NEAREST`  
`MLIB_BILINEAR`  
`MLIB_BICUBIC`  
`MLIB_BICUBIC2`

*edge* Type of edge condition. It can be one of the following:

`MLIB_EDGE_DST_NO_WRITE`  
`MLIB_EDGE_DST_FILL_ZERO`  
`MLIB_EDGE_OP_NEAREST`  
`MLIB_EDGE_SRC_EXTEND`  
`MLIB_EDGE_SRC_PADDED`

*colormap* Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_ImageZoom(3MLIB)`, `mllib_ImageZoom_Fp(3MLIB)`, `mllib_ImageZoomIn2X(3MLIB)`, `mllib_ImageZoomIn2X_Fp(3MLIB)`, `mllib_ImageZoomIndex(3MLIB)`, `mllib_ImageZoomOut2X(3MLIB)`, `mllib_ImageZoomOut2X_Fp(3MLIB)`, `mllib_ImageZoomOut2XIndex(3MLIB)`, `attributes(5)`

**Name** mllib\_ImageZoomIndex – zoom on color-indexed image

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomIndex(mllib_image *dst, const mllib_image *src,
 mllib_d64 zoomx, mllib_d64 zoomy, mllib_filter filter, mllib_edge edge,
 const void *colormap);
```

**Description** The `mllib_ImageZoomIndex()` function will enlarge or minify the source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*            Pointer to destination image.

*src*            Pointer to source image.

*zoomx*          X zoom factor.  $zoomx > 0.0$ .

*zoomy*          Y zoom factor.  $zoomy > 0.0$ .

*filter*          Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge*           Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

*colormap*       Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoom\(3MLIB\)](#), [mllib\\_ImageZoom\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomIn2X\(3MLIB\)](#), [mllib\\_ImageZoomIn2X\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomIn2XIndex\(3MLIB\)](#), [mllib\\_ImageZoomOut2X\(3MLIB\)](#), [mllib\\_ImageZoomOut2X\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomOut2XIndex\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_ImageZoomOut2X – 0.5X zoom

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_ImageZoomOut2X(mlib_image *dst, const mlib_image *src,
 mlib_filter filter, mlib_edge edge);
```

**Description** The `mlib_ImageZoomOut2X()` function minifies the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*filter* Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge* Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_ImageZoom(3MLIB)`, `mlib_ImageZoom_Fp(3MLIB)`, `mlib_ImageZoomIn2X(3MLIB)`, `mlib_ImageZoomIn2X_Fp(3MLIB)`, `mlib_ImageZoomIn2XIndex(3MLIB)`, `mlib_ImageZoomIndex(3MLIB)`, `mlib_ImageZoomOut2X_Fp(3MLIB)`, `mlib_ImageZoomOut2XIndex(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageZoomOut2X\_Fp – 0.5X zoom

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomOut2X_Fp(mllib_image *dst,
 const mllib_image *src, mllib_filter filter, mllib_edge edge);
```

**Description** The `mllib_ImageZoomOut2X_Fp()` function minifies the floating-point source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst*        Pointer to destination image.

*src*        Pointer to source image.

*filter*     Type of resampling filter. It can be one of the following:

`MLIB_NEAREST`  
            `MLIB_BILINEAR`  
            `MLIB_BICUBIC`  
            `MLIB_BICUBIC2`

*edge*       Type of edge condition. It can be one of the following:

`MLIB_EDGE_DST_NO_WRITE`  
            `MLIB_EDGE_DST_FILL_ZERO`  
            `MLIB_EDGE_OP_NEAREST`  
            `MLIB_EDGE_SRC_EXTEND`  
            `MLIB_EDGE_SRC_PADDED`

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_ImageZoom(3MLIB)`, `mllib_ImageZoom_Fp(3MLIB)`, `mllib_ImageZoomIn2X(3MLIB)`,  
`mllib_ImageZoomIn2X_Fp(3MLIB)`, `mllib_ImageZoomIn2XIndex(3MLIB)`,  
`mllib_ImageZoomIndex(3MLIB)`, `mllib_ImageZoomOut2X(3MLIB)`,  
`mllib_ImageZoomOut2XIndex(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_ImageZoomOut2XIndex – 0.5X zoom

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageZoomOut2XIndex(mllib_image *dst,
 const mllib_image *src, mllib_filter filter, mllib_edge edge,
 const void *colormap);
```

**Description** The `mllib_ImageZoomOut2XIndex()` function minifies the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters** The function takes the following arguments:

*dst* Pointer to destination image.

*src* Pointer to source image.

*filter* Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge* Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

*colormap* Internal data structure for inverse color mapping. This data structure is generated by the `mllib_ImageColorTrue2IndexInit()` function.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_ImageZoom(3MLIB), mllib_ImageZoom_Fp(3MLIB), mllib_ImageZoomIn2X(3MLIB), mllib_ImageZoomIn2X_Fp(3MLIB), mllib_ImageZoomIn2XIndex(3MLIB), mllib_ImageZoomIndex(3MLIB), mllib_ImageZoomOut2X(3MLIB), mllib_ImageZoomOut2X_Fp(3MLIB), attributes(5)`

**Name** mllib\_ImageZoomTranslate – zoom, with translation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslate(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, mllib_filter filter,
 mllib_edge edge);
```

**Description** The `mllib_ImageZoomTranslate()` function will enlarge or minify the source image by the X and Y zoom factors, with translation. It uses the interpolation method as described by the resampling filter.

It uses the following equation for coordinate mapping:

$$\begin{aligned} x_d &= \text{zoomx} \cdot x_s + t_x \\ y_d &= \text{zoomy} \cdot y_s + t_y \end{aligned}$$

where a point with coordinates  $(x_s, y_s)$  in the source image is mapped to a point with coordinates  $(x_d, y_d)$  in the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to source image.

*zoomx*   X zoom factor. `zoomx > 0`.

*zoomy*   Y zoom factor. `zoomy > 0`.

*tx*      X translation.

*ty*      Y translation.

*filter*   Type of resampling filter. It can be one of the following:

```
MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2
```

*edge*    Type of edge condition. It can be one of the following:

MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_OP\_NEAREST  
MLIB\_EDGE\_SRC\_EXTEND  
MLIB\_EDGE\_SRC\_EXTEND\_INDEF  
MLIB\_EDGE\_SRC\_PADDED

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoomTranslate\\_Fp\(3MLIB\)](#), [mllib\\_ImageAffine\(3MLIB\)](#),  
[mllib\\_ImageAffine\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomTranslateBlend – image scaling with alpha blending

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslateBlend(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, mllib_filter filter, mllib_edge edge,
 mllib_blend blend, mllib_s32 alpha, mllib_s32 cmask);
```

**Description** The `mllib_ImageZoomTranslateBlend()` function will enlarge or minify the source image by the X and Y zoom factors, with translation, and blend it with the destination image.

It uses the following equation for coordinate mapping:

```
xd = zoomx*xs + tx
yd = zoomy*ys + ty
```

where a point with coordinates ( $x_s$ ,  $y_s$ ) in the source image is mapped to a point with coordinates ( $x_d$ ,  $y_d$ ) in the destination image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The alpha blending is closely combined with the interpolation to achieve better performance. Part of alpha blending has to be performed before or together with the interpolation if the source image has an alpha channel. In that case, the color components of each neighboring source pixel which participates in the interpolation (`src_r` and etc.) have to be pre-multiplied by the alpha component of the same source pixel (`src_a`). After the interpolation, the overall alpha (*alpha*), the interpolated source alpha (`interp_a`) and the destination pixel's original alpha (`dst_a`, if any) are used to blend the interpolated source pixel (with components `interp_r` and etc.) with the destination pixel (with components `dst_r` and etc.).

The `MLIB_BLEND_GTK_SRC` blending is similar to the SRC rule of the Porter-Duff rules for image compositing. It is defined by

```
Cd = Cs
Ad = As
```

in general, and by the following formula for this function:

```
if (interp_a != 0.0) {
 if (dst_has_alpha) {
 dst_r = interp_r/interp_a;
 dst_g = interp_g/interp_a;
 dst_b = interp_b/interp_a;
 dst_a = interp_a;
 } else {
 dst_r = interp_r;
 dst_g = interp_g;
 dst_b = interp_b;
```

```
 dst_a = 1.0; // implied
 }
} else {
 dst_r = 0;
 dst_g = 0;
 dst_b = 0;
 dst_a = 0;
}
```

The `MLIB_BLEND_GTK_SRC_OVER` or `MLIB_BLEND_GTK_SRC_OVER2` blending is similar to the `SRC_OVER` rule of the Porter-Duff rules for image compositing. It is defined by

```
Cd = Cs + Cd*(1 - As)
Ad = As + Ad*(1 - As)
```

in general, and by the following formula for this function:

```
w = alpha*interp_a + (1 - alpha*interp_a)*dst_a;
if (w != 0.0) {
 dst_r = (alpha*interp_r +
 (1 - alpha*interp_a)*dst_a*dst_r)/w;
 dst_g = (alpha*interp_g +
 (1 - alpha*interp_a)*dst_a*dst_g)/w;
 dst_b = (alpha*interp_b +
 (1 - alpha*interp_a)*dst_a*dst_b)/w;
 dst_a = w;
} else if (MLIB_BLEND_GTK_SRC_OVER) {
 dst_r = 0;
 dst_g = 0;
 dst_b = 0;
 dst_a = 0;
}
```

where *alpha*, *src\_a*, *interp\_a* and *dst\_a* are assumed to be in the range of `[0.0, 1.0]`.

For an image with 4 channels, the first or the fourth channel is considered the alpha channel if *cmask* equals 8 or 1, respectively. An image with 3 channels is considered to have no alpha channel, which is equivalent to having an alpha channel filled with all 1.0, or 0xff in case of `MLIB_BYTE`, if the general formulas for blending shown above are used.

Both *src* and *dst* must be of type `MLIB_BYTE`. They can have either 3 or 4 channels.

The *src* image cannot have width or height larger than 32767.

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to first source image.

*zoomx*    X zoom factor. *zoomx* > 0.0.



*zoomy* Y zoom factor. *zoomy* > 0.0.

*tx* X translation.

*ty* Y translation.

*filter* Type of resampling filter. It can be one of the following:

MLIB\_NEAREST  
MLIB\_BILINEAR  
MLIB\_BICUBIC  
MLIB\_BICUBIC2

*edge* Type of edge condition. It can be one of the following:

MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_OP\_NEAREST  
MLIB\_EDGE\_SRC\_EXTEND  
MLIB\_EDGE\_SRC\_EXTEND\_INDEF  
MLIB\_EDGE\_SRC\_PADDED

*blend* Type of alpha blending. It can be one of the following:

MLIB\_BLEND\_GTK\_SRC  
MLIB\_BLEND\_GTK\_SRC\_OVER  
MLIB\_BLEND\_GTK\_SRC\_OVER2

*alpha* Overall alpha for blending.

*cmask* Channel mask to indicate the alpha channel.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoomBlend\(3MLIB\)](#), [mllib\\_ImageZoomTranslateTableBlend\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomTranslate\_Fp – zoom, with translation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslate_Fp(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, mllib_filter filter,
 mllib_edge edge);
```

**Description** The `mllib_ImageZoomTranslate_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors, with translation. It uses the interpolation method as described by the resampling filter.

It uses the following equation for coordinate mapping:

$$\begin{aligned}x_d &= \text{zoomx} * x_s + t_x \\ y_d &= \text{zoomy} * y_s + t_y\end{aligned}$$

where a point with coordinates  $(x_s, y_s)$  in the source image is mapped to a point with coordinates  $(x_d, y_d)$  in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image.

*src*      Pointer to source image.

*zoomx*    X zoom factor. `zoomx > 0`.

*zoomy*    Y zoom factor. `zoomy > 0`.

*tx*        X translation.

*ty*        Y translation.

*filter*    Type of resampling filter. It can be one of the following:

`MLIB_NEAREST`  
`MLIB_BILINEAR`  
`MLIB_BICUBIC`  
`MLIB_BICUBIC2`

*edge*      Type of edge condition. It can be one of the following:

MLIB\_EDGE\_DST\_NO\_WRITE  
MLIB\_EDGE\_DST\_FILL\_ZERO  
MLIB\_EDGE\_OP\_NEAREST  
MLIB\_EDGE\_SRC\_EXTEND  
MLIB\_EDGE\_SRC\_EXTEND\_INDEF  
MLIB\_EDGE\_SRC\_PADDED

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoomTranslate\(3MLIB\)](#), [mllib\\_ImageAffine\(3MLIB\)](#),  
[mllib\\_ImageAffine\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomTranslateTable – zoom, with translation, with table-driven interpolation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_ImageZoomTranslateTable(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, const void *interp_table,
 mllib_edge edge);
```

**Description** The `mllib_ImageZoomTranslateTable()` function will enlarge or minify the source image by the X and Y zoom factors, with translation. It uses a table, *interp\_table*, to do interpolation.

It uses the following equation for coordinate mapping:

```
xd = zoomx*xs + tx
yd = zoomy*ys + ty
```

where a point with coordinates (xs, ys) in the source image is mapped to a point with coordinates (xd, yd) in the destination image.

The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

**Parameters** The function takes the following arguments:

|                     |                                                                                                                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>          | Pointer to destination image.                                                                                                                                                                                                                                                                        |
| <i>src</i>          | Pointer to source image.                                                                                                                                                                                                                                                                             |
| <i>zoomx</i>        | X zoom factor. <code>zoomx &gt; 0</code> .                                                                                                                                                                                                                                                           |
| <i>zoomy</i>        | Y zoom factor. <code>zoomy &gt; 0</code> .                                                                                                                                                                                                                                                           |
| <i>tx</i>           | X translation.                                                                                                                                                                                                                                                                                       |
| <i>ty</i>           | Y translation.                                                                                                                                                                                                                                                                                       |
| <i>interp_table</i> | Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.                                                                                                                                                                                 |
| <i>edge</i>         | Type of edge condition. It can be one of the following:<br><br><code>MLIB_EDGE_DST_NO_WRITE</code><br><code>MLIB_EDGE_DST_FILL_ZERO</code><br><code>MLIB_EDGE_OP_NEAREST</code><br><code>MLIB_EDGE_SRC_EXTEND</code><br><code>MLIB_EDGE_SRC_EXTEND_INDEF</code><br><code>MLIB_EDGE_SRC_PADDED</code> |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageInterpTableCreate\(3MLIB\)](#), [mllib\\_ImageInterpTableDelete\(3MLIB\)](#), [mllib\\_ImageZoomTranslateTable\\_Fp\(3MLIB\)](#), [mllib\\_ImageZoomTranslate\(3MLIB\)](#), [mllib\\_ImageZoomTranslate\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomTranslateTableBlend – image scaling using interpolation table, combined with alpha blending

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslateTableBlend(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, const void *table, mllib_edge edge,
 mllib_blend blend, mllib_s32 cmask);
```

**Description** The `mllib_ImageZoomTranslateTableBlend()` function will enlarge or minify the source image by the X and Y zoom factors, with translation, and blend it with the destination image.

It uses the following equation for coordinate mapping:

```
xd = zoomx*xs + tx
yd = zoomy*ys + ty
```

where a point with coordinates (xs, ys) in the source image is mapped to a point with coordinates (xd, yd) in the destination image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

It is assumed that the overall alpha for controlling the blending between the source image and the destination image has been pre-multiplied to the interpolation table for better performance.

The alpha blending is closely combined with the interpolation to achieve better performance. Part of alpha blending has to be performed before or together with the interpolation if the source image has an alpha channel. In that case, the color components of each neighboring source pixel which participates in the interpolation (src\_r and etc.) have to be pre-multiplied by the alpha component of the same source pixel (src\_a). After the interpolation, the interpolated alpha (interp\_a, which has been multiplied by the overall alpha because of the pre-multiplied interpolation table) and the destination pixel's original alpha (dst\_a, if any) are used to blend the interpolated source pixel (with components interp\_r and etc.) with the destination pixel (with components dst\_r and etc.).

The `MLIB_BLEND_GTK_SRC` blending is similar to the SRC rule of the Porter-Duff rules for image compositing. It is defined by

```
Cd = Cs
Ad = As
```

in general, and by the following formula for this function:

```
if (interp_a != 0.0) {
 if (dst_has_alpha) {
 dst_r = interp_r/interp_a;
 dst_g = interp_g/interp_a;
```

```

 dst_b = interp_b/interp_a;
 dst_a = interp_a;
 } else {
 dst_r = interp_r;
 dst_g = interp_g;
 dst_b = interp_b;
 dst_a = 1.0; // implied
 }
} else {
 dst_r = 0;
 dst_g = 0;
 dst_b = 0;
 dst_a = 0;
}

```

The `MLIB_BLEND_GTK_SRC_OVER` or `MLIB_BLEND_GTK_SRC_OVER2` blending is similar to the `SRC_OVER` rule of the Porter-Duff rules for image compositing. It is defined by

$$C_d = C_s + C_d * (1 - A_s)$$

$$A_d = A_s + A_d * (1 - A_s)$$

in general, and by the following formula for this function:

```

w = interp_a + (1 - interp_a)*dst_a;
if (w != 0.0) {
 dst_r = (interp_r + (1 - interp_a)*dst_a*dst_r)/w;
 dst_g = (interp_g + (1 - interp_a)*dst_a*dst_g)/w;
 dst_b = (interp_b + (1 - interp_a)*dst_a*dst_b)/w;
 dst_a = w;
} else if (MLIB_BLEND_GTK_SRC_OVER) {
 dst_r = 0;
 dst_g = 0;
 dst_b = 0;
 dst_a = 0;
}

```

where `src_a`, `interp_a` and `dst_a` are assumed to be in the range of `[0.0, 1.0]`.

For an image with 4 channels, the first or the fourth channel is considered the alpha channel if `cmask` equals 8 or 1, respectively. An image with 3 channels is considered to have no alpha channel, which is equivalent to having an alpha channel filled with all 1.0, or 0xff in case of `MLIB_BYTE`, if the general formulas for blending shown above are used.

Both *src* and *dst* must be of type `MLIB_BYTE`. They can have either 3 or 4 channels.

The *src* image cannot have width or height larger than 32767.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination image.
- src* Pointer to first source image.
- zoomx* X zoom factor. *zoomx* > 0.0.
- zoomy* Y zoom factor. *zoomy* > 0.0.
- tx* X translation.
- ty* Y translation.
- table* Pointer to interpolation table structure.
- edge* Type of edge condition. It can be one of the following:
  - MLIB\_EDGE\_DST\_NO\_WRITE
  - MLIB\_EDGE\_DST\_FILL\_ZERO
  - MLIB\_EDGE\_OP\_NEAREST
  - MLIB\_EDGE\_SRC\_EXTEND
  - MLIB\_EDGE\_SRC\_EXTEND\_INDEF
  - MLIB\_EDGE\_SRC\_PADDED
- blend* Type of alpha blending. It can be one of the following:
  - MLIB\_BLEND\_GTK\_SRC
  - MLIB\_BLEND\_GTK\_SRC\_OVER
  - MLIB\_BLEND\_GTK\_SRC\_OVER2
- cmask* Channel mask to indicate the alpha channel.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageZoomBlend\(3MLIB\)](#), [mllib\\_ImageZoomTranslateBlend\(3MLIB\)](#), [mllib\\_ImageInterpTableCreate\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_ImageZoomTranslateTable\_Fp – zoom, with translation, with table-driven interpolation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslateTable_Fp(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, const void *interp_table,
 mllib_edge edge);
```

**Description** The `mllib_ImageZoomTranslateTable_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors, with translation. It uses a table, *interp\_table*, to do interpolation.

It uses the following equation for coordinate mapping:

$$\begin{aligned} x_d &= \text{zoomx} \cdot x_s + t_x \\ y_d &= \text{zoomy} \cdot y_s + t_y \end{aligned}$$

where a point with coordinates ( $x_s$ ,  $y_s$ ) in the source image is mapped to a point with coordinates ( $x_d$ ,  $y_d$ ) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at  $(0.5, 0.5)$ .

**Parameters** The function takes the following arguments:

|                     |                                                                                                                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>          | Pointer to destination image.                                                                                                                                                                                        |
| <i>src</i>          | Pointer to source image.                                                                                                                                                                                             |
| <i>zoomx</i>        | X zoom factor. $\text{zoomx} > 0$ .                                                                                                                                                                                  |
| <i>zoomy</i>        | Y zoom factor. $\text{zoomy} > 0$ .                                                                                                                                                                                  |
| <i>tx</i>           | X translation.                                                                                                                                                                                                       |
| <i>ty</i>           | Y translation.                                                                                                                                                                                                       |
| <i>interp_table</i> | Pointer to an interpolation table. The table is created by the <code>mllib_ImageInterpTableCreate()</code> function.                                                                                                 |
| <i>edge</i>         | Type of edge condition. It can be one of the following:<br><br><code>MLIB_EDGE_DST_NO_WRITE</code><br><code>MLIB_EDGE_DST_FILL_ZERO</code><br><code>MLIB_EDGE_OP_NEAREST</code><br><code>MLIB_EDGE_SRC_EXTEND</code> |

MLIB\_EDGE\_SRC\_EXTEND\_INDEF  
MLIB\_EDGE\_SRC\_PADDED

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_ImageInterpTableCreate\(3MLIB\)](#), [mllib\\_ImageInterpTableDelete\(3MLIB\)](#),  
[mllib\\_ImageZoomTranslateTable\(3MLIB\)](#), [mllib\\_ImageZoomTranslate\(3MLIB\)](#),  
[mllib\\_ImageZoomTranslate\\_Fp\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_ImageZoomTranslateToGray – zoom, with translation, and convert to grayscale

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_ImageZoomTranslateToGray(mllib_image *dst,
 const mllib_image *src, mllib_d64 zoomx, mllib_d64 zoomy,
 mllib_d64 tx, mllib_d64 ty, mllib_filter filter, mllib_edge edge,
 const mllib_s32 *ghigh, const mllib_s32 *glow);
```

**Description** The `mllib_ImageZoomTranslateToGray()` function will enlarge or minify the source binary image by the X and Y zoom factors, with translation, and convert the resulting image into a grayscale image.

It uses the following equation for coordinate mapping:

$$\begin{aligned} x_d &= \text{zoomx} * x_s + tx \\ y_d &= \text{zoomy} * y_s + ty \end{aligned}$$

where a point with coordinates ( $x_s$ ,  $y_s$ ) in the source image is mapped to a point with coordinates ( $x_d$ ,  $y_d$ ) in the destination image.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination image. It must be of type `MLIB_BYTE` and have just one channel.

*src*      Pointer to source image. It must be of type `MLIB_BIT` and have just one channel.

*zoomx*    X zoom factor.  $\text{zoomx} > 0$ .

*zoomy*    Y zoom factor.  $\text{zoomy} > 0$ .

*tx*        X translation.

*ty*        Y translation.

*filter*    Type of resampling filter. It must be `MLIB_NEAREST`.

*edge*      Type of edge condition. It can be one of the following:

```
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED
```

*ghigh*    Pointer to value for '1' pixels in source image.

*glow*      Pointer to value for '0' pixels in source image.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_ImageSubsampleBinaryToGray\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_malloc – allocate a block of bytes

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
void *mlib_malloc(size_t size);
```

**Description** The `mlib_malloc()` function allocates *size* bytes on a 16-byte aligned boundary and returns a pointer to the allocated block.

This function is equivalent to `memalign(16, size)`.

**Parameters** The function takes the following arguments:

*size*      Size of the block in bytes.

**Return Values** The function returns a pointer to the allocated block if successful. Otherwise it returns a null pointer.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_free\(3MLIB\)](#), [mlib\\_realloc\(3MLIB\)](#), [malloc\(3C\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixAddS\_U8\_Mod, mllib\_MatrixAddS\_U8\_Sat, mllib\_MatrixAddS\_U8C\_Mod, mllib\_MatrixAddS\_U8C\_Sat, mllib\_MatrixAddS\_S8\_Mod, mllib\_MatrixAddS\_S8\_Sat, mllib\_MatrixAddS\_S8C\_Mod, mllib\_MatrixAddS\_S8C\_Sat, mllib\_MatrixAddS\_S16\_Mod, mllib\_MatrixAddS\_S16\_Sat, mllib\_MatrixAddS\_S16C\_Mod, mllib\_MatrixAddS\_S16C\_Sat, mllib\_MatrixAddS\_S32\_Mod, mllib\_MatrixAddS\_S32\_Sat, mllib\_MatrixAddS\_S32C\_Mod, mllib\_MatrixAddS\_S32C\_Sat – matrix addition to scalar, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixAddS_U8_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_U8_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_U8C_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_U8C_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S8_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S8_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S8C_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S8C_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32C_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

```
mllib_status mllib_MatrixAddS_S32C_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place addition of a scalar value to a matrix.

For real data, the following equation is used:

$$xz[i] = c[0] + xz[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} xz[2*i] &= c[0] + xz[2*i] \\ xz[2*i + 1] &= c[1] + xz[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the source and the destination matrix.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in the matrices.
- n* Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixAddS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixAddS\_U8\_U8\_Mod, mllib\_MatrixAddS\_U8\_U8\_Sat, mllib\_MatrixAddS\_U8C\_U8C\_Mod, mllib\_MatrixAddS\_U8C\_U8C\_Sat, mllib\_MatrixAddS\_S8\_S8\_Mod, mllib\_MatrixAddS\_S8\_S8\_Sat, mllib\_MatrixAddS\_S8C\_S8C\_Mod, mllib\_MatrixAddS\_S8C\_S8C\_Sat, mllib\_MatrixAddS\_S16\_U8\_Mod, mllib\_MatrixAddS\_S16\_U8\_Sat, mllib\_MatrixAddS\_S16\_S8\_Mod, mllib\_MatrixAddS\_S16\_S8\_Sat, mllib\_MatrixAddS\_S16\_S16\_Mod, mllib\_MatrixAddS\_S16\_S16\_Sat, mllib\_MatrixAddS\_S16C\_U8C\_Mod, mllib\_MatrixAddS\_S16C\_U8C\_Sat, mllib\_MatrixAddS\_S16C\_S8C\_Mod, mllib\_MatrixAddS\_S16C\_S8C\_Sat, mllib\_MatrixAddS\_S16C\_S16C\_Mod, mllib\_MatrixAddS\_S16C\_S16C\_Sat, mllib\_MatrixAddS\_S32\_S16\_Mod, mllib\_MatrixAddS\_S32\_S16\_Sat, mllib\_MatrixAddS\_S32\_S32\_Mod, mllib\_MatrixAddS\_S32\_S32\_Sat, mllib\_MatrixAddS\_S32C\_S16C\_Mod, mllib\_MatrixAddS\_S32C\_S16C\_Sat, mllib\_MatrixAddS\_S32C\_S32C\_Mod, mllib\_MatrixAddS\_S32C\_S32C\_Sat – matrix addition to scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixAddS_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixAddS_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);
```



```

mllib_status mllib_MatrixAddS_S16_S8_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16_S16_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16_S16_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_U8C_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_S8C_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_S16C_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S16C_S16C_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_S16_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_S16_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_S32_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32_S32_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32C_S16C_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32C_S16C_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32C_S32C_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAddS_S32C_S32C_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n);

```

**Description** Each of these functions adds a scalar value to a matrix.

For real data, the following equation is used:

$$z[i] = c[0] + x[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= c[0] + x[2*i] \\ z[2*i + 1] &= c[1] + x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the destination matrix.
- x* Pointer to the source matrix.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in the matrices.
- n* Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixAddS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixAdd\_U8\_Mod, mllib\_MatrixAdd\_U8\_Sat, mllib\_MatrixAdd\_U8C\_Mod, mllib\_MatrixAdd\_U8C\_Sat, mllib\_MatrixAdd\_S8\_Mod, mllib\_MatrixAdd\_S8\_Sat, mllib\_MatrixAdd\_S8C\_Mod, mllib\_MatrixAdd\_S8C\_Sat, mllib\_MatrixAdd\_S16\_Mod, mllib\_MatrixAdd\_S16\_Sat, mllib\_MatrixAdd\_S16C\_Mod, mllib\_MatrixAdd\_S16C\_Sat, mllib\_MatrixAdd\_S32\_Mod, mllib\_MatrixAdd\_S32\_Sat, mllib\_MatrixAdd\_S32C\_Mod, mllib\_MatrixAdd\_S32C\_Sat – matrix addition, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mllib_MatrixAdd_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAdd_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

```

```
mllib_status mllib_MatrixAdd_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place addition of the second source matrix to the first source matrix.

It uses the following equation:

$$xz[i] = xz[i] + y[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- xz*     Pointer to the first source and destination matrix.
- y*     Pointer to the second source matrix.
- m*     Number of rows in the matrices.
- n*     Number of columns in the matrices.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_MatrixAdd\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixAdd\_U8\_U8\_Mod, mlib\_MatrixAdd\_U8\_U8\_Sat,  
 mlib\_MatrixAdd\_U8C\_U8C\_Mod, mlib\_MatrixAdd\_U8C\_U8C\_Sat,  
 mlib\_MatrixAdd\_S8\_S8\_Mod, mlib\_MatrixAdd\_S8\_S8\_Sat,  
 mlib\_MatrixAdd\_S8C\_S8C\_Mod, mlib\_MatrixAdd\_S8C\_S8C\_Sat,  
 mlib\_MatrixAdd\_S16\_U8\_Mod, mlib\_MatrixAdd\_S16\_U8\_Sat,  
 mlib\_MatrixAdd\_S16\_S8\_Mod, mlib\_MatrixAdd\_S16\_S8\_Sat,  
 mlib\_MatrixAdd\_S16\_S16\_Mod, mlib\_MatrixAdd\_S16\_S16\_Sat,  
 mlib\_MatrixAdd\_S16C\_U8C\_Mod, mlib\_MatrixAdd\_S16C\_U8C\_Sat,  
 mlib\_MatrixAdd\_S16C\_S8C\_Mod, mlib\_MatrixAdd\_S16C\_S8C\_Sat,  
 mlib\_MatrixAdd\_S16C\_S16C\_Mod, mlib\_MatrixAdd\_S16C\_S16C\_Sat,  
 mlib\_MatrixAdd\_S32\_S16\_Mod, mlib\_MatrixAdd\_S32\_S16\_Sat,  
 mlib\_MatrixAdd\_S32\_S32\_Mod, mlib\_MatrixAdd\_S32\_S32\_Sat,  
 mlib\_MatrixAdd\_S32C\_S16C\_Mod, mlib\_MatrixAdd\_S32C\_S16C\_Sat,  
 mlib\_MatrixAdd\_S32C\_S32C\_Mod, mlib\_MatrixAdd\_S32C\_S32C\_Sat – matrix addition

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
 #include <mlib.h>

```

mlib_status mlib_MatrixAdd_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixAdd_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

```

```
mllib_status mlib_MatrixAdd_S16_S16_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16_S16_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_U8C_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_S8C_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_S16C_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S16C_S16C_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32_S16_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32_S16_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32_S32_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32_S32_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32C_S16C_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32C_S16C_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32C_S32C_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixAdd_S32C_S32C_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions adds the first source matrix to the second source matrix and writes the output to the destination matrix.

It uses the following equation:

$$z[i] = x[i] + y[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the destination matrix.
- x*     Pointer to the first source matrix.
- y*     Pointer to the second source matrix.
- m*     Number of rows in the matrices.
- n*     Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixAdd\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixAve\_U8, mllib\_MatrixAve\_U8C, mllib\_MatrixAve\_S8, mllib\_MatrixAve\_S8C, mllib\_MatrixAve\_S16, mllib\_MatrixAve\_S16C, mllib\_MatrixAve\_S32, mllib\_MatrixAve\_S32C  
– matrix average, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixAve_U8(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_U8C(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S8(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S8C(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16C(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32C(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place averaging of two matrices.

It uses the following equation:

$$xz[i] = (xz[i] + y[i] + 1) / 2$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- xz*    Pointer to the first source and destination matrix.
- y*     Pointer to the second source matrix.
- m*     Number of rows in the matrices.
- n*     Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.



**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_MatrixAve\\_U8\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixAve\_U8\_U8, mllib\_MatrixAve\_U8C\_U8C, mllib\_MatrixAve\_S8\_S8, mllib\_MatrixAve\_S8C\_S8C, mllib\_MatrixAve\_S16\_U8, mllib\_MatrixAve\_S16\_S8, mllib\_MatrixAve\_S16\_S16, mllib\_MatrixAve\_S16C\_U8C, mllib\_MatrixAve\_S16C\_S8C, mllib\_MatrixAve\_S16C\_S16C, mllib\_MatrixAve\_S32\_S16, mllib\_MatrixAve\_S32\_S32, mllib\_MatrixAve\_S32C\_S16C, mllib\_MatrixAve\_S32C\_S32C – matrix average

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_MatrixAve_U8_U8(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_U8C_U8C(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S8_S8(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S8C_S8C(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16_U8(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16_S8(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16_S16(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16C_U8C(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16C_S8C(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S16C_S16C(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32_S16(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32_S32(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32C_S16C(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixAve_S32C_S32C(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions computes the average of the first source matrix and the second source matrix and writes the output to the destination matrix.

It uses the following equation:

$$z[i] = (x[i] + y[i] + 1) / 2$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the destination matrix.
- $x$  Pointer to the first source matrix.
- $y$  Pointer to the second source matrix.
- $m$  Number of rows in the matrices.
- $n$  Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixAve\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixMaximumMag\_U8C, mllib\_MatrixMaximumMag\_S8C, mllib\_MatrixMaximumMag\_S16C, mllib\_MatrixMaximumMag\_S32C, mllib\_MatrixMaximumMag\_F32C, mllib\_MatrixMaximumMag\_D64C – find the first element with the maximum magnitude in a matrix

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_MatrixMaximumMag_U8C(mllib_u8 *max, const mllib_u8 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMaximumMag_S8C(mllib_s8 *max, const mllib_s8 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMaximumMag_S16C(mllib_s16 *max, const mllib_s16 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMaximumMag_S32C(mllib_s32 *max, const mllib_s32 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMaximumMag_F32C(mllib_f32 *max, const mllib_f32 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMaximumMag_D64C(mllib_d64 *max, const mllib_d64 *x,
 mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions finds the first element with the maximum magnitude in a complex matrix, then puts the real and imaginary parts of it into max[0] and max[1], respectively.

**Parameters** Each of the functions takes the following arguments:

*max*     Pointer to the first element with the maximum magnitude

*x*        Pointer to the first element of the source matrix.

*m*        Number of rows in the source matrix

*n*        Number of columns in the source matrix

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_MatrixMinimumMag\\_U8C\(3MLIB\)](#), [mllib\\_VectorMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_VectorMinimumMag\\_U8C\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixMaximum\_U8, mlib\_MatrixMaximum\_S8, mlib\_MatrixMaximum\_S16, mlib\_MatrixMaximum\_S32, mlib\_MatrixMaximum\_F32, mlib\_MatrixMaximum\_D64 – find the maximum value in a matrix

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_MatrixMaximum_U8(mlib_u8 *max, const mlib_u8 *x,
 mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMaximum_S8(mlib_s8 *max, const mlib_s8 *x,
 mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMaximum_S16(mlib_s16 *max, const mlib_s16 *x,
 mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMaximum_S32(mlib_s32 *max, const mlib_s32 *x,
 mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMaximum_F32(mlib_f32 *max, const mlib_f32 *x,
 mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMaximum_D64(mlib_d64 *max, const mlib_d64 *x,
 mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions finds the maximum value of all elements in a matrix.

It uses the following equation:

$$\max[0] = \text{MAX}\{ x[i] \mid i = 0, 1, \dots, (m*n - 1) \}$$

**Parameters** Each of the functions takes the following arguments:

*max*     Pointer to the maximum value.  
*x*        Pointer to the first element of the source matrix.  
*m*        Number of rows in the source matrix.  
*n*        Number of columns in the source matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_MatrixMinimum_U8(3MLIB)`, `mllib_VectorMaximum_U8(3MLIB)`,  
`mllib_VectorMinimum_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_MatrixMinimumMag\_U8C, mllib\_MatrixMinimumMag\_S8C, mllib\_MatrixMinimumMag\_S16C, mllib\_MatrixMinimumMag\_S32C, mllib\_MatrixMinimumMag\_F32C, mllib\_MatrixMinimumMag\_D64C – find the first element with the minimum magnitude in a matrix

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_MatrixMinimumMag_U8C(mllib_u8 *min, const mllib_u8 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMinimumMag_S8C(mllib_s8 *min, const mllib_s8 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMinimumMag_S16C(mllib_s16 *min, const mllib_s16 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMinimumMag_S32C(mllib_s32 *min, const mllib_s32 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMinimumMag_F32C(mllib_f32 *min, const mllib_f32 *x,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMinimumMag_D64C(mllib_d64 *min, const mllib_d64 *x,
 mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions finds the first element with the minimum magnitude in a complex matrix, then puts the real and imaginary parts of it into `min[0]` and `min[1]`, respectively.

**Parameters** Each of the functions takes the following arguments:

*min*     Pointer to the first element with the minimum magnitude.  
*x*        Pointer to the first element of the source matrix.  
*m*        Number of rows in the source matrix.  
*n*        Number of columns in the source matrix.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_VectorMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_VectorMinimumMag\\_U8C\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixMinimum\_U8, mllib\_MatrixMinimum\_S8, mllib\_MatrixMinimum\_S16, mllib\_MatrixMinimum\_S32, mllib\_MatrixMinimum\_F32, mllib\_MatrixMinimum\_D64 – find the minimum value in a matrix

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_MatrixMinimum_U8(mllib_u8 *min, const mllib_u8 *x,  
 mllib_s32 m, mllib_s32 n);  
  
mllib_status mllib_MatrixMinimum_S8(mllib_s8 *min, const mllib_s8 *x,  
 mllib_s32 m, mllib_s32 n);  
  
mllib_status mllib_MatrixMinimum_S16(mllib_s16 *min, const mllib_s16 *x,  
 mllib_s32 m, mllib_s32 n);  
  
mllib_status mllib_MatrixMinimum_S32(mllib_s32 *min, const mllib_s32 *x,  
 mllib_s32 m, mllib_s32 n);  
  
mllib_status mllib_MatrixMinimum_F32(mllib_f32 *min, const mllib_f32 *x,  
 mllib_s32 m, mllib_s32 n);  
  
mllib_status mllib_MatrixMinimum_D64(mllib_d64 *min, const mllib_d64 *x,  
 mllib_s32 m, mllib_s32 n);`

**Description** Each of these functions finds the minimum value of all elements in a matrix.

It uses the following equation:

$$\min[0] = \text{MIN}\{ x[i] \mid i = 0, 1, \dots, (m \cdot n - 1) \}$$

**Parameters** Each of the functions takes the following arguments:

- min*     Pointer to the minimum value.
- x*       Pointer to the first element of the source matrix.
- m*       Number of rows in the source matrix.
- n*       Number of columns in the source matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** `mlib_MatrixMaximum_U8(3MLIB)`, `mlib_VectorMaximum_U8(3MLIB)`,  
`mlib_VectorMinimum_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_MatrixMulShift\_S16\_S16\_Mod, mllib\_MatrixMulShift\_S16\_S16\_Sat, mllib\_MatrixMulShift\_S16C\_S16C\_Mod, mllib\_MatrixMulShift\_S16C\_S16C\_Sat – matrix multiplication plus shifting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixMulShift_S16_S16_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m,
 mllib_s32 l, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulShift_S16_S16_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m,
 mllib_s32 l, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulShift_S16C_S16C_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m,
 mllib_s32 l, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulShift_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 m,
 mllib_s32 l, mllib_s32 n, mllib_s32 shift);
```

**Description** Each of these functions performs a multiplication of two matrices and shifts the result.

For real data, the following equation is used:

$$z[i*n + j] = \left\{ \sum_{k=0}^{l-1} (x[i*l + k] * y[k*n + j]) \right\} * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (m - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*(i*n + j)] = \left\{ \sum_{k=0}^{l-1} (xR*yR - xI*yI) \right\} * 2^{**(-shift)}$$

$$z[2*(i*n + j) + 1] = \left\{ \sum_{k=0}^{l-1} (xR*yI + xI*yR) \right\} * 2^{**(-shift)}$$

where

```
xR = x[2*(i*l + k)]
xI = x[2*(i*l + k) + 1]
yR = y[2*(k*n + j)]
yI = y[2*(k*n + j) + 1]
i = 0, 1, ..., (m - 1)
j = 0, 1, ..., (n - 1)
```

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the result matrix, in row major order.
- x* Pointer to the first element of the first matrix, in row major order.
- y* Pointer to the first element of the second matrix, in row major order.
- m* Number of rows in the first matrix.  $m > 0$ .
- l* Number of columns in the first matrix, and the number of rows in the second matrix.  $l > 0$ .
- n* Number of columns in the second matrix.  $n > 0$ .
- shift* Right shifting factor.  $1 \leq \text{shift} \leq 16$ .

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMul\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)



## REFERENCE

### Multimedia Library Functions - Part 4

**Name** mllib\_MatrixMulSShift\_U8\_Mod, mllib\_MatrixMulSShift\_U8\_Sat, mllib\_MatrixMulSShift\_U8C\_Mod, mllib\_MatrixMulSShift\_U8C\_Sat, mllib\_MatrixMulSShift\_S8\_Mod, mllib\_MatrixMulSShift\_S8\_Sat, mllib\_MatrixMulSShift\_S8C\_Mod, mllib\_MatrixMulSShift\_S8C\_Sat, mllib\_MatrixMulSShift\_S16\_Mod, mllib\_MatrixMulSShift\_S16\_Sat, mllib\_MatrixMulSShift\_S16C\_Mod, mllib\_MatrixMulSShift\_S16C\_Sat, mllib\_MatrixMulSShift\_S32\_Mod, mllib\_MatrixMulSShift\_S32\_Sat, mllib\_MatrixMulSShift\_S32C\_Mod, mllib\_MatrixMulSShift\_S32C\_Sat – matrix multiplication by scalar plus shifting, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mllib_MatrixMulSShift_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 m, mllib_s32 n, mllib_s32 shift);

```

```

mlib_status mlib_MatrixMulSShift_S32C_Mod(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S32C_Sat(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

```

**Description** Each of these functions performs an in-place multiplication of a matrix with a scalar and shifts the result.

For real data, the following equation is used:

$$xz[i] = c[0] * xz[i] * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned}
 tmp &= xz[2*i] \\
 xz[2*i] &= (c[0]*tmp - c[1]*xz[2*i + 1]) * 2^{**(-shift)} \\
 xz[2*i + 1] &= (c[1]*tmp + c[0]*xz[2*i + 1]) * 2^{**(-shift)}
 \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

The ranges of valid shift are:

$1 \leq \text{shift} \leq 8$  for U8, S8, U8C, S8C types  
 $1 \leq \text{shift} \leq 16$  for S16, S16C types  
 $1 \leq \text{shift} \leq 31$  for S32, S32C types

**Parameters** Each of the functions takes the following arguments:

*xz* Pointer to the source and destination matrix.  
*c* Pointer to the source scalar. When the function is used with complex data types,  $c[0]$  contains the scalar for the real part, and  $c[1]$  contains the scalar for the imaginary part.  
*m* Number of rows in each matrix.  
*n* Number of columns in each matrix.  
*shift* Right shifting factor.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMulSShift\\_U8\\_U8\\_Mod\(3MLIB\), attributes\(5\)](#)



**Name** mllib\_MatrixMulSShift\_U8\_U8\_Mod, mllib\_MatrixMulSShift\_U8\_U8\_Sat, mllib\_MatrixMulSShift\_U8C\_U8C\_Mod, mllib\_MatrixMulSShift\_U8C\_U8C\_Sat, mllib\_MatrixMulSShift\_S8\_S8\_Mod, mllib\_MatrixMulSShift\_S8\_S8\_Sat, mllib\_MatrixMulSShift\_S8C\_S8C\_Mod, mllib\_MatrixMulSShift\_S8C\_S8C\_Sat, mllib\_MatrixMulSShift\_S16\_S16\_Mod, mllib\_MatrixMulSShift\_S16\_S16\_Sat, mllib\_MatrixMulSShift\_S16C\_S16C\_Mod, mllib\_MatrixMulSShift\_S16C\_S16C\_Sat, mllib\_MatrixMulSShift\_S32\_S32\_Mod, mllib\_MatrixMulSShift\_S32\_S32\_Sat, mllib\_MatrixMulSShift\_S32C\_S32C\_Mod, mllib\_MatrixMulSShift\_S32C\_S32C\_Sat – matrix multiplication by scalar plus shifting

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_MatrixMulSShift_U8_U8_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8_U8_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8C_U8C_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_U8C_U8C_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8_S8_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8_S8_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8C_S8C_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16_S16_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16_S16_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);
```

```
mllib_status mllib_MatrixMulSShift_S16C_S16C_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32_S32_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32_S32_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);

mllib_status mllib_MatrixMulSShift_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 m, mllib_s32 n,
 mllib_s32 shift);
```

**Description** Each of these functions performs a multiplication of a matrix with a scalar and shifts the result.

For real data, the following equation is used:

$$z[i] = c[0] * x[i] * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= (c[0] * x[2*i] - c[1] * x[2*i + 1]) * 2^{**(-shift)} \\ z[2*i + 1] &= (c[1] * x[2*i] + c[0] * x[2*i + 1]) * 2^{**(-shift)} \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

The ranges of valid shift are:

$1 \leq \text{shift} \leq 8$  for U8, S8, U8C, S8C types

$1 \leq \text{shift} \leq 16$  for S16, S16C types

$1 \leq \text{shift} \leq 31$  for S32, S32C types

**Parameters** Each of the functions takes the following arguments:

$z$  Pointer to the destination matrix.

$x$  Pointer to the source matrix.

- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in each matrix.
- n* Number of columns in each matrix.
- shift* Right shifting factor.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMulSShift\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixMulS\_U8\_Mod, mllib\_MatrixMulS\_U8\_Sat, mllib\_MatrixMulS\_U8C\_Mod, mllib\_MatrixMulS\_U8C\_Sat, mllib\_MatrixMulS\_S8\_Mod, mllib\_MatrixMulS\_S8\_Sat, mllib\_MatrixMulS\_S8C\_Mod, mllib\_MatrixMulS\_S8C\_Sat, mllib\_MatrixMulS\_S16\_Mod, mllib\_MatrixMulS\_S16\_Sat, mllib\_MatrixMulS\_S16C\_Mod, mllib\_MatrixMulS\_S16C\_Sat, mllib\_MatrixMulS\_S32\_Mod, mllib\_MatrixMulS\_S32\_Sat, mllib\_MatrixMulS\_S32C\_Mod, mllib\_MatrixMulS\_S32C\_Sat – matrix multiplication by scalar, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixMulS_U8_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_U8_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_U8C_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_U8C_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S8_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S8_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S8C_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S8C_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S16_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S16_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S16C_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S16C_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S32_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S32_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixMulS_S32C_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

```
mllib_status mllib_MatrixMulS_S32C_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place multiplication of a scalar to a matrix.

For real data, the following equation is used:

$$xz[i] = c[0]*xz[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} tmp &= xz[2*i] \\ xz[2*i] &= c[0]*tmp - c[1]*xz[2*i + 1] \\ xz[2*i + 1] &= c[1]*tmp + c[0]*xz[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the source and destination matrix.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in each matrix.
- n* Number of columns in each matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMulS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixMulS\_U8\_U8\_Mod, mlib\_MatrixMulS\_U8\_U8\_Sat, mlib\_MatrixMulS\_U8C\_U8C\_Mod, mlib\_MatrixMulS\_U8C\_U8C\_Sat, mlib\_MatrixMulS\_S8\_S8\_Mod, mlib\_MatrixMulS\_S8\_S8\_Sat, mlib\_MatrixMulS\_S8C\_S8C\_Mod, mlib\_MatrixMulS\_S8C\_S8C\_Sat, mlib\_MatrixMulS\_S16\_U8\_Mod, mlib\_MatrixMulS\_S16\_U8\_Sat, mlib\_MatrixMulS\_S16\_S8\_Mod, mlib\_MatrixMulS\_S16\_S8\_Sat, mlib\_MatrixMulS\_S16\_S16\_Mod, mlib\_MatrixMulS\_S16\_S16\_Sat, mlib\_MatrixMulS\_S16C\_U8C\_Mod, mlib\_MatrixMulS\_S16C\_U8C\_Sat, mlib\_MatrixMulS\_S16C\_S8C\_Mod, mlib\_MatrixMulS\_S16C\_S8C\_Sat, mlib\_MatrixMulS\_S16C\_S16C\_Mod, mlib\_MatrixMulS\_S16C\_S16C\_Sat, mlib\_MatrixMulS\_S32\_S16\_Mod, mlib\_MatrixMulS\_S32\_S16\_Sat, mlib\_MatrixMulS\_S32\_S32\_Mod, mlib\_MatrixMulS\_S32\_S32\_Sat, mlib\_MatrixMulS\_S32C\_S16C\_Mod, mlib\_MatrixMulS\_S32C\_S16C\_Sat, mlib\_MatrixMulS\_S32C\_S32C\_Mod, mlib\_MatrixMulS\_S32C\_S32C\_Sat – matrix multiplication by scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mlib_status mlib_MatrixMulS_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
```

```

mllib_status mllib_MatrixMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

```

**Description** Each of these functions multiplies a matrix by a scalar.

For real data, the following equation is used:

$$z[i] = c[0] * x[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= c[0]*x[2*i] - c[1]*x[2*i + 1] \\ z[2*i + 1] &= c[1]*x[2*i] + c[0]*x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the destination matrix.
- x*     Pointer to the source matrix.
- c*     Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m*     Number of rows in each matrix.
- n*     Number of columns in each matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMulS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_MatrixMul\_U8\_U8\_Mod, mllib\_MatrixMul\_U8\_U8\_Sat, mllib\_MatrixMul\_U8C\_U8C\_Mod, mllib\_MatrixMul\_U8C\_U8C\_Sat, mllib\_MatrixMul\_S8\_S8\_Mod, mllib\_MatrixMul\_S8\_S8\_Sat, mllib\_MatrixMul\_S8C\_S8C\_Mod, mllib\_MatrixMul\_S8C\_S8C\_Sat, mllib\_MatrixMul\_S16\_U8\_Mod, mllib\_MatrixMul\_S16\_U8\_Sat, mllib\_MatrixMul\_S16\_S8\_Mod, mllib\_MatrixMul\_S16\_S8\_Sat, mllib\_MatrixMul\_S16\_S16\_Mod, mllib\_MatrixMul\_S16\_S16\_Sat, mllib\_MatrixMul\_S16C\_U8C\_Mod, mllib\_MatrixMul\_S16C\_U8C\_Sat, mllib\_MatrixMul\_S16C\_S8C\_Mod, mllib\_MatrixMul\_S16C\_S8C\_Sat, mllib\_MatrixMul\_S16C\_S16C\_Mod, mllib\_MatrixMul\_S16C\_S16C\_Sat, mllib\_MatrixMul\_S32\_S16\_Mod, mllib\_MatrixMul\_S32\_S16\_Sat, mllib\_MatrixMul\_S32\_S32\_Mod, mllib\_MatrixMul\_S32\_S32\_Sat, mllib\_MatrixMul\_S32C\_S16C\_Mod, mllib\_MatrixMul\_S32C\_S16C\_Sat, mllib\_MatrixMul\_S32C\_S32C\_Mod, mllib\_MatrixMul\_S32C\_S32C\_Sat – matrix multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixMul_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
```

```
mllib_status mllib_MatrixMul_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);

mllib_status mllib_MatrixMul_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
```

**Description** Each of these functions performs matrix multiplication of the first matrix to the second matrix or the first complex matrix to the second complex matrix.

For real data, the following equation is used:

$$z[i*n + j] = \sum_{k=0}^{l-1} (x[i*l + k] * y[k*n + j])$$

where  $i = 0, 1, \dots, (m - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*(i*n + j)] = \sum_{k=0}^{l-1} (xR*yR - xI*yI)$$

$$z[2*(i*n + j) + 1] = \sum_{k=0}^{l-1} (xR*yI + xI*yR)$$

where

$$\begin{aligned} xR &= x[2*(i*l + k)] \\ xI &= x[2*(i*l + k) + 1] \\ yR &= y[2*(k*n + j)] \\ yI &= y[2*(k*n + j) + 1] \\ i &= 0, 1, \dots, (m - 1) \\ j &= 0, 1, \dots, (n - 1) \end{aligned}$$

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the destination matrix.
- $x$  Pointer to the first source matrix.
- $y$  Pointer to the second source matrix.
- $m$  Number of rows in the first matrix.
- $l$  Number of columns in the first matrix, and number of rows in the second matrix.
- $n$  Number of columns in the second matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixMulShift\\_S16\\_S16\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixScale\_U8\_Mod, mllib\_MatrixScale\_U8\_Sat, mllib\_MatrixScale\_U8C\_Mod, mllib\_MatrixScale\_U8C\_Sat, mllib\_MatrixScale\_S8\_Mod, mllib\_MatrixScale\_S8\_Sat, mllib\_MatrixScale\_S8C\_Mod, mllib\_MatrixScale\_S8C\_Sat, mllib\_MatrixScale\_S16\_Mod, mllib\_MatrixScale\_S16\_Sat, mllib\_MatrixScale\_S16C\_Mod, mllib\_MatrixScale\_S16C\_Sat, mllib\_MatrixScale\_S32\_Mod, mllib\_MatrixScale\_S32\_Sat, mllib\_MatrixScale\_S32C\_Mod, mllib\_MatrixScale\_S32C\_Sat – matrix linear scaling, in place

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_MatrixScale_U8_Mod(mllib_u8 *xz, const mllib_u8 *a,
 const mllib_u8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_U8_Sat(mllib_u8 *xz, const mllib_u8 *a,
 const mllib_u8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_U8C_Mod(mllib_u8 *xz, const mllib_u8 *a,
 const mllib_u8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_U8C_Sat(mllib_u8 *xz, const mllib_u8 *a,
 const mllib_u8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S8_Mod(mllib_s8 *xz, const mllib_s8 *a,
 const mllib_s8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S8_Sat(mllib_s8 *xz, const mllib_s8 *a,
 const mllib_s8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S8C_Mod(mllib_s8 *xz, const mllib_s8 *a,
 const mllib_s8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S8C_Sat(mllib_s8 *xz, const mllib_s8 *a,
 const mllib_s8 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S16_Mod(mllib_s16 *xz, const mllib_s16 *a,
 const mllib_s16 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S16_Sat(mllib_s16 *xz, const mllib_s16 *a,
 const mllib_s16 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S16C_Mod(mllib_s16 *xz, const mllib_s16 *a,
 const mllib_s16 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S16C_Sat(mllib_s16 *xz, const mllib_s16 *a,
 const mllib_s16 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S32_Mod(mllib_s32 *xz, const mllib_s32 *a,
 const mllib_s32 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S32_Sat(mllib_s32 *xz, const mllib_s32 *a,
 const mllib_s32 *b, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixScale_S32C_Mod(mllib_s32 *xz, const mllib_s32 *a,
 const mllib_s32 *b, mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S32C_Sat(mllib_s32 *xz, const mllib_s32 *a,
 const mllib_s32 *b, mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place multiplication of a matrix by a scalar and then adds an offset.

For real data, the following equation is used:

$$xz[i] = a[0]*xz[i] + b[0]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} tmp &= xz[2*i] \\ xz[2*i] &= a[0]*tmp - a[1]*xz[2*i + 1] + b[0] \\ xz[2*i + 1] &= a[1]*tmp + a[0]*xz[2*i + 1] + b[1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the source and destination matrix.
- a* Pointer to the source scaling factor. When the function is used with complex data types, *a*[0] contains the scalar for the real part, and *a*[1] contains the scalar for the imaginary part.
- b* Pointer to the source offset. When the function is used with complex data types, *b*[0] contains the offset for the real part, and *b*[1] contains the offset for the imaginary part.
- m* Number of rows in the matrix.
- n* Number of columns in the matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixScale\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixScale\_U8\_U8\_Mod, mllib\_MatrixScale\_U8\_U8\_Sat, mllib\_MatrixScale\_U8C\_U8C\_Mod, mllib\_MatrixScale\_U8C\_U8C\_Sat, mllib\_MatrixScale\_S8\_S8\_Mod, mllib\_MatrixScale\_S8\_S8\_Sat, mllib\_MatrixScale\_S8C\_S8C\_Mod, mllib\_MatrixScale\_S8C\_S8C\_Sat, mllib\_MatrixScale\_S16\_U8\_Mod, mllib\_MatrixScale\_S16\_U8\_Sat, mllib\_MatrixScale\_S16\_S8\_Mod, mllib\_MatrixScale\_S16\_S8\_Sat, mllib\_MatrixScale\_S16\_S16\_Mod, mllib\_MatrixScale\_S16\_S16\_Sat, mllib\_MatrixScale\_S16C\_U8C\_Mod, mllib\_MatrixScale\_S16C\_U8C\_Sat, mllib\_MatrixScale\_S16C\_S8C\_Mod, mllib\_MatrixScale\_S16C\_S8C\_Sat, mllib\_MatrixScale\_S16C\_S16C\_Mod, mllib\_MatrixScale\_S16C\_S16C\_Sat, mllib\_MatrixScale\_S32\_S16\_Mod, mllib\_MatrixScale\_S32\_S16\_Sat, mllib\_MatrixScale\_S32\_S32\_Mod, mllib\_MatrixScale\_S32\_S32\_Sat, mllib\_MatrixScale\_S32C\_S16C\_Mod, mllib\_MatrixScale\_S32C\_S16C\_Sat, mllib\_MatrixScale\_S32C\_S32C\_Mod, mllib\_MatrixScale\_S32C\_S32C\_Sat – matrix linear scaling

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixScale_U8_U8_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_U8_U8_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_U8C_U8C_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_U8C_U8C_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S8_S8_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S8_S8_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S8C_S8C_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_MatrixScale_S16_U8_Mod(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16_U8_Sat(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16_S8_Mod(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16_S8_Sat(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16_S16_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16_S16_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_U8C_Mod(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_U8C_Sat(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *a, const mllib_u8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_S8C_Mod(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_S8C_Sat(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *a, const mllib_s8 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_S16C_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32_S16_Mod(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32_S16_Sat(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);
```



```

mllib_status mllib_MatrixScale_S32_S32_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *a, const mllib_s32 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32_S32_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *a, const mllib_s32 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32C_S16C_Mod(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32C_S16C_Sat(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *a, const mllib_s16 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *a, const mllib_s32 *b,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixScale_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *a, const mllib_s32 *b,
 mllib_s32 m, mllib_s32 n);

```

**Description** Each of these functions multiplies a matrix by a scalar and then adds an offset.

For real data, the following equation is used:

$$z[i] = a[0]*x[i] + b[0]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= a[0]*x[2*i] - a[1]*x[2*i + 1] + b[0] \\ z[2*i + 1] &= a[1]*x[2*i] + a[0]*x[2*i + 1] + b[1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the destination matrix.
- x*     Pointer to the source matrix.
- a*     Pointer to the source scaling factor. When the function is used with complex data types, *a*[0] contains the scalar for the real part, and *a*[1] contains the scalar for the imaginary part.
- b*     Pointer to the source offset. When the function is used with complex data types, *b*[0] contains the offset for the real part, and *b*[1] contains the offset for the imaginary part.
- m*     Number of rows in each matrix.
- n*     Number of columns in each matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixScale\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixSubS\_U8\_Mod, mllib\_MatrixSubS\_U8\_Sat, mllib\_MatrixSubS\_U8C\_Mod, mllib\_MatrixSubS\_U8C\_Sat, mllib\_MatrixSubS\_S8\_Mod, mllib\_MatrixSubS\_S8\_Sat, mllib\_MatrixSubS\_S8C\_Mod, mllib\_MatrixSubS\_S8C\_Sat, mllib\_MatrixSubS\_S16\_Mod, mllib\_MatrixSubS\_S16\_Sat, mllib\_MatrixSubS\_S16C\_Mod, mllib\_MatrixSubS\_S16C\_Sat, mllib\_MatrixSubS\_S32\_Mod, mllib\_MatrixSubS\_S32\_Sat, mllib\_MatrixSubS\_S32C\_Mod, mllib\_MatrixSubS\_S32C\_Sat – matrix subtraction from scalar, in place

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_MatrixSubS_U8_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_U8_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_U8C_Mod(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_U8C_Sat(mllib_u8 *xz, const mllib_u8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S8_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S8_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S8C_Mod(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S8C_Sat(mllib_s8 *xz, const mllib_s8 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S16_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S16_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S16C_Mod(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S16C_Sat(mllib_s16 *xz, const mllib_s16 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S32_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S32_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSubS_S32C_Mod(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

```
mllib_status mllib_MatrixSubS_S32C_Sat(mllib_s32 *xz, const mllib_s32 *c,
 mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place subtraction of a matrix from a scalar.

For real data, the following equation is used:

$$xz[i] = c[0] - xz[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} xz[2*i] &= c[0] - xz[2*i] \\ xz[2*i + 1] &= c[1] - xz[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the source and destination matrix.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in the matrices.
- n* Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixSubS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixSubS\_U8\_U8\_Mod, mllib\_MatrixSubS\_U8\_U8\_Sat, mllib\_MatrixSubS\_U8C\_U8C\_Mod, mllib\_MatrixSubS\_U8C\_U8C\_Sat, mllib\_MatrixSubS\_S8\_S8\_Mod, mllib\_MatrixSubS\_S8\_S8\_Sat, mllib\_MatrixSubS\_S8C\_S8C\_Mod, mllib\_MatrixSubS\_S8C\_S8C\_Sat, mllib\_MatrixSubS\_S16\_U8\_Mod, mllib\_MatrixSubS\_S16\_U8\_Sat, mllib\_MatrixSubS\_S16\_S8\_Mod, mllib\_MatrixSubS\_S16\_S8\_Sat, mllib\_MatrixSubS\_S16\_S16\_Mod, mllib\_MatrixSubS\_S16\_S16\_Sat, mllib\_MatrixSubS\_S16C\_U8C\_Mod, mllib\_MatrixSubS\_S16C\_U8C\_Sat, mllib\_MatrixSubS\_S16C\_S8C\_Mod, mllib\_MatrixSubS\_S16C\_S8C\_Sat, mllib\_MatrixSubS\_S16C\_S16C\_Mod, mllib\_MatrixSubS\_S16C\_S16C\_Sat, mllib\_MatrixSubS\_S32\_S16\_Mod, mllib\_MatrixSubS\_S32\_S16\_Sat, mllib\_MatrixSubS\_S32\_S32\_Mod, mllib\_MatrixSubS\_S32\_S32\_Sat, mllib\_MatrixSubS\_S32C\_S16C\_Mod, mllib\_MatrixSubS\_S32C\_S16C\_Sat, mllib\_MatrixSubS\_S32C\_S32C\_Mod, mllib\_MatrixSubS\_S32C\_S32C\_Sat – matrix subtraction from scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixSubS_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSubS_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 m, mlib_s32 n);
```

```
mllib_status mlib_MatrixSubS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSubS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions subtracts a matrix from a scalar.

For real data, the following equation is used:

$$z[i] = c[0] - x[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= c[0] - x[2*i] \\ z[2*i + 1] &= c[1] - x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m*n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the destination matrix.
- x* Pointer to the source matrix.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- m* Number of rows in the matrices.
- n* Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixSubS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_MatrixSub\_U8\_Mod, mllib\_MatrixSub\_U8\_Sat, mllib\_MatrixSub\_U8C\_Mod, mllib\_MatrixSub\_U8C\_Sat, mllib\_MatrixSub\_S8\_Mod, mllib\_MatrixSub\_S8\_Sat, mllib\_MatrixSub\_S8C\_Mod, mllib\_MatrixSub\_S8C\_Sat, mllib\_MatrixSub\_S16\_Mod, mllib\_MatrixSub\_S16\_Sat, mllib\_MatrixSub\_S16C\_Mod, mllib\_MatrixSub\_S16C\_Sat, mllib\_MatrixSub\_S32\_Mod, mllib\_MatrixSub\_S32\_Sat, mllib\_MatrixSub\_S32C\_Mod, mllib\_MatrixSub\_S32C\_Sat – matrix subtraction, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_MatrixSub_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);

mllib_status mllib_MatrixSub_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);
```



```
mllib_status mllib_MatrixSub_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 m, mllib_s32 n);
```

**Description** Each of these functions performs an in-place subtraction of the second matrix from the first matrix.

It uses the following equation:

$$xz[i] = xz[i] - y[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first source and destination matrix.
- y* Pointer to the second source matrix.
- m* Number of rows in the matrices.
- n* Number of columns in the matrices.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixSub\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixSub\_U8\_U8\_Mod, mlib\_MatrixSub\_U8\_U8\_Sat,  
 mlib\_MatrixSub\_U8C\_U8C\_Mod, mlib\_MatrixSub\_U8C\_U8C\_Sat,  
 mlib\_MatrixSub\_S8\_S8\_Mod, mlib\_MatrixSub\_S8\_S8\_Sat,  
 mlib\_MatrixSub\_S8C\_S8C\_Mod, mlib\_MatrixSub\_S8C\_S8C\_Sat,  
 mlib\_MatrixSub\_S16\_U8\_Mod, mlib\_MatrixSub\_S16\_U8\_Sat,  
 mlib\_MatrixSub\_S16\_S8\_Mod, mlib\_MatrixSub\_S16\_S8\_Sat,  
 mlib\_MatrixSub\_S16\_S16\_Mod, mlib\_MatrixSub\_S16\_S16\_Sat,  
 mlib\_MatrixSub\_S16C\_U8C\_Mod, mlib\_MatrixSub\_S16C\_U8C\_Sat,  
 mlib\_MatrixSub\_S16C\_S8C\_Mod, mlib\_MatrixSub\_S16C\_S8C\_Sat,  
 mlib\_MatrixSub\_S16C\_S16C\_Mod, mlib\_MatrixSub\_S16C\_S16C\_Sat,  
 mlib\_MatrixSub\_S32\_S16\_Mod, mlib\_MatrixSub\_S32\_S16\_Sat,  
 mlib\_MatrixSub\_S32\_S32\_Mod, mlib\_MatrixSub\_S32\_S32\_Sat,  
 mlib\_MatrixSub\_S32C\_S16C\_Mod, mlib\_MatrixSub\_S32C\_S16C\_Sat,  
 mlib\_MatrixSub\_S32C\_S32C\_Mod, mlib\_MatrixSub\_S32C\_S32C\_Sat – matrix subtraction

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mlib_MatrixSub_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mlib_MatrixSub_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

```

```

mllib_status mllib_MatrixSub_S16_S16_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16_S16_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_U8C_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_S8C_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_S16C_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S16C_S16C_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32_S16_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32_S16_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32_S32_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32_S32_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32C_S16C_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32C_S16C_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32C_S32C_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_MatrixSub_S32C_S32C_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

```

**Description** Each of these functions subtracts the second matrix from the first matrix.

It uses the following equation:

$$z[i] = x[i] - y[i]$$

where  $i = 0, 1, \dots, (m*n - 1)$  for real data;  $i = 0, 1, \dots, (m*n*2 - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the destination matrix.
- x*     Pointer to the first source matrix.
- y*     Pointer to the second source matrix.
- m*     Number of rows in the matrices.
- n*     Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixSub\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixTranspose\_U8, mlib\_MatrixTranspose\_U8C, mlib\_MatrixTranspose\_S8, mlib\_MatrixTranspose\_S8C, mlib\_MatrixTranspose\_S16, mlib\_MatrixTranspose\_S16C, mlib\_MatrixTranspose\_S32, mlib\_MatrixTranspose\_S32C – matrix transpose, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_MatrixTranspose_U8(mlib_u8 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_U8C(mlib_u8 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S8(mlib_s8 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S8C(mlib_s8 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S16(mlib_s16 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S16C(mlib_s16 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S32(mlib_s32 *xz, mlib_s32 mn);
mlib_status mlib_MatrixTranspose_S32C(mlib_s32 *xz, mlib_s32 mn);
```

**Description** Each of these functions performs an in-place transpose of a square matrix.

For real data, the following pseudo code applies:

```
for (i = 1; i < mn; i++) {
 for (j = 0; j < i; i++) {
 tmp = xz[i*mn + j];
 xz[i*mn + j] = xz[j*mn + i];
 xz[j*mn + i] = tmp;
 }
}
```

For complex data, the following pseudo code applies:

```
for (i = 1; i < mn; i++) {
 for (j = 0; j < i; i++) {
 tmp0 = xz[2*(i*mn + j)];
 tmp1 = xz[2*(i*mn + j) + 1];
 xz[2*(i*mn + j)] = xz[2*(j*mn + i)];
 xz[2*(i*mn + j) + 1] = xz[2*(j*mn + i) + 1];
 xz[2*(j*mn + i)] = tmp0;
 xz[2*(j*mn + i) + 1] = tmp1;
 }
}
```

**Parameters** Each of the functions takes the following arguments:

*xz*     Pointer to the source and destination matrix.  
*mn*     Number of rows and columns in the matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_MatrixTranspose\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_MatrixTranspose\_U8\_U8, mlib\_MatrixTranspose\_U8C\_U8C, mlib\_MatrixTranspose\_S8\_S8, mlib\_MatrixTranspose\_S8C\_S8C, mlib\_MatrixTranspose\_S16\_S16, mlib\_MatrixTranspose\_S16C\_S16C, mlib\_MatrixTranspose\_S32\_S32, mlib\_MatrixTranspose\_S32C\_S32C – matrix transpose

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_MatrixTranspose_U8_U8(mlib_u8 *z,
 const mlib_u8 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_U8C_U8C(mlib_u8 *z,
 const mlib_u8 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S8_S8(mlib_s8 *z,
 const mlib_s8 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S8C_S8C(mlib_s8 *z,
 const mlib_s8 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S16_S16(mlib_s16 *z,
 const mlib_s16 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S16C_S16C(mlib_s16 *z,
 const mlib_s16 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S32_S32(mlib_s32 *z,
 const mlib_s32 *x, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixTranspose_S32C_S32C(mlib_s32 *z,
 const mlib_s32 *x, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions computes the transpose of the input matrix.

For real data, the following equation is used:

$$z[j*m + i] = x[i*n + j]$$

where  $i = 0, 1, \dots, (m - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*(j*m + i)] &= x[2*(i*n + j)] \\ z[2*(j*m + i) + 1] &= x[2*(i*n + j) + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (m - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the destination matrix. The output data type must be the same as the input data type.
- $x$  Pointer to the source matrix.
- $m$  Number of rows in the source matrix.

*n*     Number of columns in the source matrix.

**Return Values**   Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**   See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**   [mllib\\_MatrixTranspose\\_U8\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_MatrixUnit\_U8, mllib\_MatrixUnit\_U8C, mllib\_MatrixUnit\_S8, mllib\_MatrixUnit\_S8C, mllib\_MatrixUnit\_S16, mllib\_MatrixUnit\_S16C, mllib\_MatrixUnit\_S32, mllib\_MatrixUnit\_S32C – Unit matrix generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_MatrixUnit_U8(mllib_u8 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_U8C(mllib_u8 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S8(mllib_s8 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S8C(mllib_s8 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S16(mllib_s16 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S16C(mllib_s16 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S32(mllib_s32 *z, mllib_s32 n);
mllib_status mllib_MatrixUnit_S32C(mllib_s32 *z, mllib_s32 n);
```

**Description** Each of these functions sets the values for a unit matrix.

For real data, the following equation is used:

$$z[i*n + j] = 1 \quad \text{if } i == j$$

$$z[i*n + j] = 0 \quad \text{if } i \neq j$$

where  $i = 0, 1, \dots, (n - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*(i*n + j)] = 1 \quad \text{if } i == j$$

$$z[2*(i*n + j)] = 0 \quad \text{if } i \neq j$$

$$z[2*(i*n + j) + 1] = 0$$

where  $i = 0, 1, \dots, (n - 1)$ ;  $j = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

$z$      Pointer to the destination matrix.

$n$      Number of rows and columns in the matrix.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_memcpy – copy a block of bytes

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void *mllib_memcpy(void *dst, const void *src, size_t n);
```

**Description** The mllib\_memcpy() function copies *n* bytes from memory area *src* to *dst*. It returns *dst*. The memory areas may not overlap. Use mllib\_memmove() if the memory areas do overlap.

This function is a wrapper of the standard C function memcpy().

**Parameters** The function takes the following arguments:

*dst*     Pointer to the destination.

*src*     Pointer to the source.

*n*       Number of bytes to be copied.

**Return Values** The function returns a pointer to the destination.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_memmove\(3MLIB\)](#), [mllib\\_memset\(3MLIB\)](#), [memory\(3C\)](#), [attributes\(5\)](#)

**Name** mllib\_memmove – copy a block of bytes

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`void *mllib_memmove(void *dst, const void *src, size_t n);`

**Description** The `mllib_memmove()` function copies *n* bytes from memory area *src* to *dst*. Copying between objects that overlap will take place correctly. It returns *dst*.

This function is a wrapper of the standard C function `memmove()`.

**Parameters** The function takes the following arguments:

*dst*      Pointer to the destination.

*src*      Pointer to the source.

*n*        Number of bytes to be copied.

**Return Values** The function returns a pointer to the destination.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_memcpy\(3MLIB\)](#), [mllib\\_memset\(3MLIB\)](#), [memory\(3C\)](#), [attributes\(5\)](#)

**Name** mlib\_memset – set a block of bytes

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
void *mlib_memset(void *s, mlib_s32 c, size_t n);
```

**Description** The `mlib_memset()` function sets the first  $n$  bytes in memory area  $s$  to the value of  $c$  (converted to an unsigned char). It returns  $s$ .

This function is a wrapper of the standard C function `memset()`.

**Parameters** The function takes the following arguments:

$s$      Pointer to the destination.

$c$      Value to set.

$n$      Number of bytes to be set.

**Return Values** The function returns a pointer to the destination.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_memcpy\(3MLIB\)](#), [mlib\\_memmove\(3MLIB\)](#), [memory\(3C\)](#), [attributes\(5\)](#)

**Name** mllib\_realloc – reallocate a block of bytes

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void *mllib_realloc(void *ptr, size_t size);
```

**Description** The `mllib_realloc()` function changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block.

This function is a wrapper of the standard C function `realloc()`.

**Parameters** The function takes the following arguments:

*size*      New size of the block in bytes.  
*ptr*      Pointer to a block.

**Return Values** The function returns a pointer to the reallocated block if successful. Otherwise it returns a null pointer.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_free\(3MLIB\)](#), [mllib\\_malloc\(3MLIB\)](#), [malloc\(3C\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalADPCM2Bits2Linear – adaptive differential pulse code modulation (ADPCM)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalADPCM2Bits2Linear(mlib_s16 *pcm,
 const mlib_u8 *adpcm, void *state, mlib_s32 n);
```

**Description** The `mlib_SignalADPCM2Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.723 or G.726 16kbps 2-bit ADPCM to 16-bit linear PCM format.

**Parameters** The function takes the following arguments:

*pcm*            Linear PCM sample array.  
*adpcm*        ADPCM code array.  
*state*         Internal structure of the codec.  
*n*             Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM3Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalADPCM3Bits2Linear – adaptive differential pulse code modulation (ADPCM)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalADPCM3Bits2Linear(mllib_s16 *pcm,
 const mllib_u8 *adpcm, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalADPCM3Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.723 or G.726 24kbps 3-bit ADPCM to 16-bit linear PCM format.

**Parameters** The function takes the following arguments:

- pcm*            Linear PCM sample array.
- adpcm*        ADPCM code array.
- state*        Internal structure of the codec.
- n*            Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalADPCM2Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM4Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM5Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCMFree\(3MLIB\)](#), [mllib\\_SignalADPCMInit\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM2Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM3Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM4Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM5Bits\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalADPCM4Bits2Linear – adaptive differential pulse code modulation (ADPCM)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalADPCM4Bits2Linear(mlib_s16 *pcm,
 const mlib_u8 *adpcm, void *state, mlib_s32 n);
```

**Description** The `mlib_SignalADPCM4Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.721 or G.726 32kbps 4-bit ADPCM to 16-bit linear PCM format.

**Parameters** The function takes the following arguments:

*pcm*            Linear PCM sample array.  
*adpcm*        ADPCM code array.  
*state*        Internal structure of the codec.  
*n*            Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM3Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalADPCM5Bits2Linear – adaptive differential pulse code modulation (ADPCM)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalADPCM5Bits2Linear(mllib_s16 *pcm,
 const mllib_u8 *adpcm, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalADPCM5Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.723 or G.726 40kbps 5-bit ADPCM to 16-bit linear PCM format.

**Parameters** The function takes the following arguments:

- pcm* Linear PCM sample array.
- adpcm* ADPCM code array.
- state* Internal structure of the codec.
- n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalADPCM2Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM3Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM4Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCMFree\(3MLIB\)](#), [mllib\\_SignalADPCMInit\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM2Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM3Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM4Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM5Bits\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalADPCMFree – adaptive differential pulse code modulation (ADPCM)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

`void mlib_SignalADPCMFree(void *state);`

**Description** The `mlib_SignalADPCMFree()` function frees the internal structure for the codec for functions that perform adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications.

**Parameters** The function takes the following arguments:

*state* Internal structure of the codec.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM3Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalADPCMInit – adaptive differential pulse code modulation (ADPCM)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_SignalADPCMInit(void **state);`

**Description** The `mllib_SignalADPCMInit()` function creates the internal structure for the codec for functions that perform adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications.

**Parameters** The function takes the following arguments:  
  
*state*      Internal structure of the codec.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalADPCM2Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM3Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM4Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM5Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCMFree\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM2Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM3Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM4Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM5Bits\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalALaw2Linear – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalALaw2Linear(mllib_s16 *pcm, const mllib_u8 *acode,
 mllib_s32 n);
```

**Description** The `mllib_SignalALaw2Linear()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

*pcm*        Linear PCM sample array.  
*acode*      A-law code array.  
*n*           Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_SignalALaw2uLaw(3MLIB)`, `mllib_SignalLinear2ALaw(3MLIB)`,  
`mllib_SignalLinear2uLaw(3MLIB)`, `mllib_SignaluLaw2ALaw(3MLIB)`,  
`mllib_SignaluLaw2Linear(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalALaw2uLaw – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalALaw2uLaw(mllib_u8 *ucode, const mllib_u8 *acode,
 mllib_s32 n);
```

**Description** The `mllib_SignalALaw2uLaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

- ucode* m-law code array.
- acode* A-law code array.
- n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalALaw2Linear\(3MLIB\)](#), [mllib\\_SignalLinear2ALaw\(3MLIB\)](#), [mllib\\_SignalLinear2uLaw\(3MLIB\)](#), [mllib\\_SignaluLaw2ALaw\(3MLIB\)](#), [mllib\\_SignaluLaw2Linear\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalAutoCorrel\_S16, mlib\_SignalAutoCorrel\_S16S, mlib\_SignalAutoCorrel\_F32, mlib\_SignalAutoCorrel\_F32S – signal auto-correlation

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalAutoCorrel_S16(mlib_d64 *correl,
 const mlib_s16 *src, mlib_s32 disp, mlib_s32 n);

mlib_status mlib_SignalAutoCorrel_S16S(mlib_d64 *correl,
 const mlib_s16 *src, mlib_s32 disp, mlib_s32 n);

mlib_status mlib_SignalAutoCorrel_F32(mlib_d64 *correl,
 const mlib_f32 *src, mlib_s32 disp, mlib_s32 n);

mlib_status mlib_SignalAutoCorrel_F32S(mlib_d64 *correl,
 const mlib_f32 *src, mlib_s32 disp, mlib_s32 n);
```

**Description** Each of these functions performs auto-correlation.

For monaural signals, the following equation is used:

$$\text{correl}[0] = \frac{1}{n-d} * \sum_{i=0}^{n-d-1} (\text{src}[i] * \text{src}[i+d])$$

For stereo signals, the following equation is used:

$$\text{correl}[0] = \frac{1}{n-d} * \sum_{i=0}^{n-d-1} (\text{src}[2*i] * \text{src}[2*(i+d)])$$

$$\text{correl}[1] = \frac{1}{n-d} * \sum_{i=0}^{n-d-1} (\text{src}[2*i+1] * \text{src}[2*(i+d)+1])$$

where  $d = \text{disp}$ .

**Parameters** Each of the functions takes the following arguments:

*correl*     Pointer to the auto-correlation array. In the stereo version, `correl[0]` contains the auto-correlation of channel 0, and `correl[1]` contains the auto-correlation of channel 1.

*src*        Source signal array.

*disp*       Displacement.  $0 \leq \text{disp} < n$ .

*n*          Number of samples in the source signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCrossCorrel\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



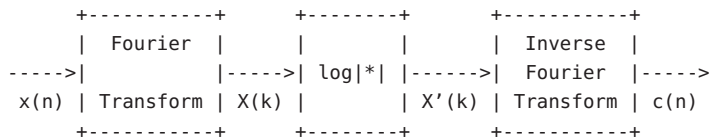
**Name** mllib\_SignalCepstral\_F32 – perform cepstral analysis

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalCepstral_F32(mllib_f32 *cepst,
 const mllib_f32 *signal, void *state);
```

**Description** The mllib\_SignalCepstral\_F32() function performs cepstral analysis.

The basic operations to compute the cepstrum is shown below.



where  $x(n)$  is the input signal and  $c(n)$  is its cepstrum. In mathematics, they are

$$X(k) = \sum_{n=0}^{N-1} x(n) * \exp(-j * \frac{2 * \pi * k * n}{N})$$

$$X'(k) = \log |X(k)|$$

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \exp(j * \frac{2 * \pi * k * n}{N})$$

Since  $X'(k)$  is real and even (symmetric), i.e.

$$X'(k) = X'(N - k)$$

the  $c(n)$  is real and the equation becomes Cosine transform.

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \cos(\frac{2 * \pi * k * n}{N})$$

The cepstral coefficients in LPC is a special case of the above.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- cepst*      The cepstral coefficients.
- signal*     The input signal vector.
- state*      Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCepstralInit\\_F32\(3MLIB\)](#), [mllib\\_SignalCepstralFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalCepstralFree\_S16, mllib\_SignalCepstralFree\_F32 – clean up for cepstral analysis

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalCepstralFree_S16(void *state);
void mllib_SignalCepstralFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for cepstral analysis.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstral\\_F32\(3MLIB\)](#),  
[mllib\\_SignalCepstral\\_S16\\_Adpt\(3MLIB\)](#), [mllib\\_SignalCepstralInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalCepstralInit\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalCepstralInit\_S16, mllib\_SignalCepstralInit\_F32 – initialization for cepstral analysis

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalCepstralInit_S16(void *state, mllib_s32 order);
mllib_status mllib_SignalCepstralInit_F32(void *state, mllib_s32 order);
```

**Description** Each of these functions initializes the internal state structure for cepstral analysis.

The init function performs internal state structure allocation and global initialization. Per function call initialization is done in each function, so the same internal state structure can be reused for multiple function calls.

**Parameters** Each of the functions takes the following arguments:

*order*     The order of the input signal vector and the cepstral coefficients, where length = 2\*\*order.  
  
*state*     Pointer to the internal state structure.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstral\\_S16\\_Adp\(3MLIB\)](#), [mllib\\_SignalCepstral\\_F32\(3MLIB\)](#), [mllib\\_SignalCepstralFree\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstralFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

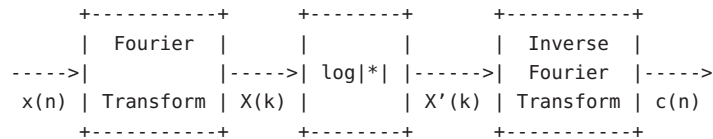
**Name** mllib\_SignalCepstral\_S16 – perform cepstral analysis

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalCepstral_S16(mllib_s16 *cepst,
 mllib_s32 cscale, const mllib_s16 *signal, void *state);
```

**Description** The `mllib_SignalCepstral_S16()` function performs cepstral analysis. The user supplied scaling factor will be used and the output will be saturated if necessary.

The basic operations to compute the cepstrum is shown below.



where  $x(n)$  is the input signal and  $c(n)$  is its cepstrum. In mathematics, they are

$$X(k) = \sum_{n=0}^{N-1} x(n) * \exp(-j * \frac{2 * \pi * k * n}{N})$$

$$X'(k) = \log |X(k)|$$

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \exp(j * \frac{2 * \pi * k * n}{N})$$

Since  $X'(k)$  is real and even (symmetric), i.e.

$$X'(k) = X'(N - k)$$

the  $c(n)$  is real and the equation becomes Cosine transform.

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \cos(\frac{2 * \pi * k * n}{N})$$

The cepstral coefficients in LPC is a special case of the above.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- cepst*      The cepstral coefficients.
- cscale*     The scaling factor of cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- signal*     The input signal vector, the signal samples are in Q15 format.
- state*      Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCepstralInit\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstral\\_S16\\_Adp\(3MLIB\)](#), [mllib\\_SignalCepstralFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

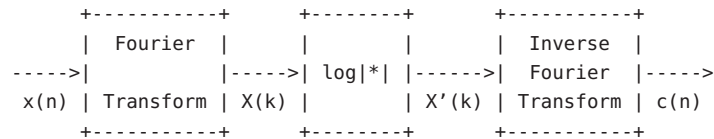
**Name** mllib\_SignalCepstral\_S16\_AdP – perform cepstral analysis

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalCepstral_S16_AdP(mllib_s16 *cepst,
 mllib_s32 *cscale, const mllib_s16 *signal, void *state);
```

**Description** The mllib\_SignalCepstral\_S16\_AdP() function performs cepstral analysis. The scaling factor of the output data will be calculated based on the actual data.

The basic operations to compute the cepstrum is shown below.



where  $x(n)$  is the input signal and  $c(n)$  is its cepstrum. In mathematics, they are

$$X(k) = \sum_{n=0}^{N-1} x(n) * \exp(-j * \frac{2*PI*k*n}{N})$$

$$X'(k) = \log|X(k)|$$

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \exp(j * \frac{2*PI*k*n}{N})$$

Since  $X'(k)$  is real and even (symmetric), i.e.

$$X'(k) = X'(N - k)$$

the  $c(n)$  is real and the equation becomes Cosine transform.

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) * \cos(\frac{2*PI*k*n}{N})$$

The cepstral coefficients in LPC is a special case of the above.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- cepst*      The cepstral coefficients.
- cscale*     The scaling factor of cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- signal*     The input signal vector, the signal samples are in Q15 format.
- state*      Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalCepstralInit\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalCepstralFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalConvertShift\_F32\_U8, mlib\_SignalConvertShift\_F32\_S8, mlib\_SignalConvertShift\_F32\_S16, mlib\_SignalConvertShift\_F32\_S32, mlib\_SignalConvertShift\_F32S\_U8S, mlib\_SignalConvertShift\_F32S\_S8S, mlib\_SignalConvertShift\_F32S\_S16S, mlib\_SignalConvertShift\_F32S\_S32S – data type convert with shifting

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalConvertShift_F32_U8(mlib_f32 *dst, const mlib_u8 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32_S8(mlib_f32 *dst, const mlib_s8 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32_S16(mlib_f32 *dst, const mlib_s16 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32_S32(mlib_f32 *dst, const mlib_s32 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32S_U8S(mlib_f32 *dst, const mlib_u8 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32S_S8S(mlib_f32 *dst, const mlib_s8 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32S_S16S(mlib_f32 *dst, const mlib_s16 *src,
 mlib_s32 shift, mlib_s32 n);
```

```
mlib_status mlib_SignalConvertShift_F32S_S32S(mlib_f32 *dst, const mlib_s32 *src,
 mlib_s32 shift, mlib_s32 n);
```

**Description** Each of these functions performs data type convert with shifting.

The following equation is used:

$$\text{dst}[i] = \text{src}[i] * 2^{**\text{shift}}$$

See the following table for available variations of this group of data type convert functions.

| Type[*] | F32 | F32S |
|---------|-----|------|
| U8      | Y   |      |
| S8      | Y   |      |
| S16     | Y   |      |
| S32     | Y   |      |
| U8S     |     | Y    |

| Type[*] | F32 | F32S |
|---------|-----|------|
| S8S     |     | Y    |
| S16S    |     | Y    |
| S32S    |     | Y    |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src* Source signal array.
- shift* Left shifting factor.
- n* Number of samples in the source signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalConvertShift\\_U8\\_S8\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalConvertShift\_U8\_S8\_Sat, mlib\_SignalConvertShift\_U8\_S16\_Sat, mlib\_SignalConvertShift\_U8\_S32\_Sat, mlib\_SignalConvertShift\_U8\_F32\_Sat, mlib\_SignalConvertShift\_U8S\_S8S\_Sat, mlib\_SignalConvertShift\_U8S\_S16S\_Sat, mlib\_SignalConvertShift\_U8S\_S32S\_Sat, mlib\_SignalConvertShift\_U8S\_F32S\_Sat, mlib\_SignalConvertShift\_S8\_U8\_Sat, mlib\_SignalConvertShift\_S8\_S16\_Sat, mlib\_SignalConvertShift\_S8\_S32\_Sat, mlib\_SignalConvertShift\_S8\_F32\_Sat, mlib\_SignalConvertShift\_S8S\_U8S\_Sat, mlib\_SignalConvertShift\_S8S\_S16S\_Sat, mlib\_SignalConvertShift\_S8S\_S32S\_Sat, mlib\_SignalConvertShift\_S8S\_F32S\_Sat, mlib\_SignalConvertShift\_S16\_U8\_Sat, mlib\_SignalConvertShift\_S16\_S8\_Sat, mlib\_SignalConvertShift\_S16\_S32\_Sat, mlib\_SignalConvertShift\_S16\_F32\_Sat, mlib\_SignalConvertShift\_S16S\_U8S\_Sat, mlib\_SignalConvertShift\_S16S\_S8S\_Sat, mlib\_SignalConvertShift\_S16S\_S32S\_Sat, mlib\_SignalConvertShift\_S16S\_F32S\_Sat, mlib\_SignalConvertShift\_S32\_U8\_Sat, mlib\_SignalConvertShift\_S32\_S8\_Sat, mlib\_SignalConvertShift\_S32\_S16\_Sat, mlib\_SignalConvertShift\_S32\_F32\_Sat, mlib\_SignalConvertShift\_S32S\_U8S\_Sat, mlib\_SignalConvertShift\_S32S\_S8S\_Sat, mlib\_SignalConvertShift\_S32S\_S16S\_Sat, mlib\_SignalConvertShift\_S32S\_F32S\_Sat – data type convert with shifting

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`

```
#include <mlib.h>
```

```
mlib_status mlib_SignalConvertShift_U8_S8_Sat(mlib_u8 *dst,
 const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_S16_Sat(mlib_u8 *dst,
 const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_S32_Sat(mlib_u8 *dst,
 const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_F32_Sat(mlib_u8 *dst,
 const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S8S_Sat(mlib_u8 *dst,
 const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S16S_Sat(mlib_u8 *dst,
 const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S32S_Sat(mlib_u8 *dst,
 const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_F32S_Sat(mlib_u8 *dst,
 const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_U8_Sat(mlib_s8 *dst,
 const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_S16_Sat(mlib_s8 *dst,
 const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);
```

```
mllib_status mllib_SignalConvertShift_S8_S32_Sat(mllib_s8 *dst,
 const mllib_s32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S8_F32_Sat(mllib_s8 *dst,
 const mllib_f32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S8S_U8S_Sat(mllib_s8 *dst,
 const mllib_u8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S8S_S16S_Sat(mllib_s8 *dst,
 const mllib_s16 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S8S_S32S_Sat(mllib_s8 *dst,
 const mllib_s32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S8S_F32S_Sat(mllib_s8 *dst,
 const mllib_f32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16_U8_Sat(mllib_s16 *dst,
 const mllib_u8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16_S8_Sat(mllib_s16 *dst,
 const mllib_s8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16_S32_Sat(mllib_s16 *dst,
 const mllib_s32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16_F32_Sat(mllib_s16 *dst,
 const mllib_f32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16S_U8S_Sat(mllib_s16 *dst,
 const mllib_u8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16S_S8S_Sat(mllib_s16 *dst,
 const mllib_s8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16S_S32S_Sat(mllib_s16 *dst,
 const mllib_s32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S16S_F32S_Sat(mllib_s16 *dst,
 const mllib_f32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S32_U8_Sat(mllib_s32 *dst,
 const mllib_u8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S32_S8_Sat(mllib_s32 *dst,
 const mllib_s8 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S32_S16_Sat(mllib_s32 *dst,
 const mllib_s16 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S32_F32_Sat(mllib_s32 *dst,
 const mllib_f32 *src, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalConvertShift_S32S_U8S_Sat(mllib_s32 *dst,
 const mllib_u8 *src, mllib_s32 shift, mllib_s32 n);
```

```

mlib_status mlib_SignalConvertShift_S32S_S8S_Sat(mlib_s32 *dst,
 const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_S16S_Sat(mlib_s32 *dst,
 const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_F32S_Sat(mlib_s32 *dst,
 const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

```

**Description** Each of these functions performs data type convert with shifting.

The following equation is used:

$$\text{dst}[i] = \text{saturate}(\text{src}[i] * 2^{\text{shift}})$$

See the following tables for available variations of this group of data type convert functions.

| Type[*] | U8 | S8 | S16 | S32 |
|---------|----|----|-----|-----|
| U8      |    | Y  | Y   | Y   |
| S8      | Y  |    | Y   | Y   |
| S16     | Y  | Y  |     | Y   |
| S32     | Y  | Y  | Y   |     |
| F32     | Y  | Y  | Y   | Y   |

| Type[*] | U8S | S8S | S16S | S32S |
|---------|-----|-----|------|------|
| U8S     |     | Y   | Y    | Y    |
| S8S     | Y   |     | Y    | Y    |
| S16S    | Y   | Y   |      | Y    |
| S32S    | Y   | Y   | Y    |      |
| F32S    | Y   | Y   | Y    | Y    |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** Each of the functions takes the following arguments:

*dst* Destination signal array.  
*src* Source signal array.  
*shift* Left shifting factor.  
*n* Number of samples in the source signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalConvertShift\\_F32\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalConv\_S16\_S16\_Sat, mllib\_SignalConv\_S16S\_S16S\_Sat,  
mllib\_SignalConv\_F32\_F32, mllib\_SignalConv\_F32S\_F32S – signal convolution

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalConv_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, mllib_s32 m,
 mllib_s32 n);

mllib_status mllib_SignalConv_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, mllib_s32 m,
 mllib_s32 n);

mllib_status mllib_SignalConv_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src1, const mllib_f32 *src2, mllib_s32 m,
 mllib_s32 n);

mllib_status mllib_SignalConv_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src1, const mllib_f32 *src2, mllib_s32 m,
 mllib_s32 n);
```

**Description** Each of these functions performs convolution.

For monaural signals, the following equation is used:

$$\text{dst}[i] = \sum_{j=0}^{m-1} (\text{src1}[j] * \text{src2}[i - j]) \quad \text{if } m \leq n$$

$$\text{dst}[i] = \sum_{j=0}^{n-1} (\text{src2}[j] * \text{src1}[i - j]) \quad \text{if } m > n$$

where  $i = 0, 1, \dots, (m + n - 2)$ .

For stereo signals, the following equation is used:

$$\text{dst}[2*i] = \sum_{j=0}^{m-1} (\text{src1}[2*j] * \text{src2}[2*(i - j)])$$

$$\text{dst}[2*i + 1] = \sum_{j=0}^{m-1} (\text{src1}[2*j + 1] * \text{src2}[2*(i - j) + 1])$$

if  $m \leq n$ , or

$$\text{dst}[2*i] = \sum_{j=0}^{n-1} (\text{src2}[2*j] * \text{src1}[2*(i - j)])$$

$$dst[2*i + 1] = \sum_{j=0}^{n-1} (src2[2*j + 1] * src1[2*(i - j) + 1])$$

if  $m > n$ ; where  $i = 0, 1, \dots, (m + n - 2)$ .

**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src1* First source signal array.
- src2* Second source signal array.
- m* Number of samples in the first source signal array.
- n* Number of samples in the second source signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalCrossCorrel\_S16, mllib\_SignalCrossCorrel\_S16S, mllib\_SignalCrossCorrel\_F32, mllib\_SignalCrossCorrel\_F32S – signal cross correlation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalCrossCorrel_S16(mlib_d64 *correl,
 const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);

mllib_status mllib_SignalCrossCorrel_S16S(mlib_d64 *correl,
 const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);

mllib_status mllib_SignalCrossCorrel_F32(mlib_d64 *correl,
 const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);

mllib_status mllib_SignalCrossCorrel_F32S(mlib_d64 *correl,
 const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);
```

**Description** Each of these functions performs cross correlation.

For monaural signals, the following equation is used:

$$\text{correl}[0] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[i] * \text{src2}[i])$$

For stereo signals, the following equation is used:

$$\text{correl}[0] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[2*i] * \text{src2}[2*i])$$

$$\text{correl}[1] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[2*i + 1] * \text{src2}[2*i + 1])$$

**Parameters** Each of the functions takes the following arguments:

*correl*     Pointer to the cross correlation array. In the stereo version, `correl[0]` contains the cross correlation of channel 0, and `correl[1]` contains the cross correlation of channel 1.

*src1*       First source signal array.

*src2*       Second source signal array.

*n*           Number of samples in the source signal arrays.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalAutoCorrel\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDownSample\_S16\_S16, mllib\_SignalDownSample\_S16S\_S16S, mllib\_SignalDownSample\_F32\_F32, mllib\_SignalDownSample\_F32S\_F32S – signal downsampling

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalDownSample_S16_S16(mllib_s16 *dst,
 const mllib_s16 *src, mllib_s32 factor, mllib_s32 phase,
 mllib_s32 n);

mllib_status mllib_SignalDownSample_S16S_S16S(mllib_s16 *dst,
 const mllib_s16 *src, mllib_s32 factor, mllib_s32 phase,
 mllib_s32 n);

mllib_status mllib_SignalDownSample_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, mllib_s32 factor, mllib_s32 phase,
 mllib_s32 n);

mllib_status mllib_SignalDownSample_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, mllib_s32 factor, mllib_s32 phase,
 mllib_s32 n);
```

**Description** Each of these functions performs downsampling.

For monaural signals, the following equation is used:

$$\text{dst}[i] = \text{src}[i * \text{factor} + \text{phase}]$$

where  $i = 0, 1, \dots, (n - 1 - \text{phase}) / \text{factor}$ .

For stereo signals, the following equation is used:

$$\begin{aligned} \text{dst}[2*i] &= \text{src}[2*(i*\text{factor} + \text{phase})] \\ \text{dst}[2*i + 1] &= \text{src}[2*(i*\text{factor} + \text{phase}) + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1 - \text{phase}) / \text{factor}$ .

**Parameters** Each of the functions takes the following arguments:

|               |                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>dst</i>    | Output signal array.                                                                                                             |
| <i>src</i>    | Input signal array.                                                                                                              |
| <i>factor</i> | Factor by which to downsample. $\text{factor} \geq 1$ .                                                                          |
| <i>phase</i>  | Parameter that determines relative position of an output value, within the input signal. $0 \leq \text{phase} < \text{factor}$ . |
| <i>n</i>      | Number of samples in the input signal array.                                                                                     |

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalUpSample\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWKScalar\_F32 – perform dynamic time warping for K-best paths on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKScalar_F32(mllib_d64 *dist,
 const mllib_f32 *dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWKScalar_F32()` function performs dynamic time warping for K-best paths on scalar data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
----------*
|p4 |p1 |p0
| | |
----------*
| |p2 |
| | |
----------* px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

#### 4. Global Path Constraints

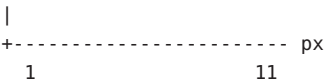
Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.





The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distances of the K-best paths.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalDTWKScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalDTWKScalarFree\_S16, mlib\_SignalDTWKScalarFree\_F32 – clean up for K-best paths of scalar data

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

`void mlib_SignalDTWKScalarFree_S16(void *state);`

`void mlib_SignalDTWKScalarFree_F32(void *state);`

**Description** Each of these functions frees the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*     Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarInit\\_F32\(3MLIB\)](#),  
[mlib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalar\\_F32\(3MLIB\)](#),  
[mlib\\_SignalDTWKScalarPath\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarPath\\_F32\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalDTWKScalarInit\_F32 – initialization for K-best paths of scalar data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalDTWKScalarInit_F32(void *state,
 const mllib_f32 *dref, mllib_s32 lref, mllib_s32 kbest,
 mllib_s32 delta, mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWKScalarInit_F32()` function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

- dref* The reference data array.
- lref* The length of the reference data array.
- kbest* The number of the best paths evaluated.
- delta* The delta in the endpoint constraints.
- local* The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.
- slope* The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.
- state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWKScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalDTWKScalarInit\_S16 – initialization for K-best paths of scalar data

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalDTWKScalarInit_S16(void *state,
 const mlib_s16 *dref, mlib_s32 lref, mlib_s32 kbest,
 mlib_s32 sref, mlib_s32 delta, mlib_s32 local,
 mlib_s32 slope);
```

**Description** The `mlib_SignalDTWKScalarInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

*dref*      The reference data array.

*lref*      The length of the reference data array.

*kbest*     The number of the best paths evaluated.

*sref*      The scaling factor of the reference data array, where `actual_data = input_data * 2**(-scaling_factor)`.

*delta*     The delta in the endpoint constraints.

*local*     The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.

*slope*     The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.

*state*     Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarPath\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWKScalarPath\_S16, mllib\_SignalDTWKScalarPath\_F32 – return K-best path on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKScalarPath_S16(mllib_s32 *path,
 mllib_s32 *lpath, mllib_s32 kpath, void *state);

mllib_status mllib_SignalDTWKScalarPath_F32(mllib_s32 *path,
 mllib_s32 *lpath, mllib_s32 kpath, void *state);
```

**Description** Each of these functions returns K-best path on scalar data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
----------*
|p4 |p1 |p0
| | |
----------*
| |p2 |
| | |
----------* px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive  $(1,0) (1,0)$  is disallowed. So path  $p4 \rightarrow p1 \rightarrow p0$  is disallowed.

#### 4. Global Path Constraints

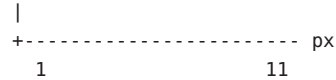
Due to local continuity constraints, certain portions of the  $(p_x, p_y)$  plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in  $(p_x, p_y)$  plane can be represented in chain code. The value of the chain code is defined as following.





The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** Each of the functions takes the following arguments:

*path* The optimal path.

*lpath* The length of the optimal path.

*kpath* The path index,  $0 \leq kpath < kbest$ .

*state* Pointer to the internal state structure.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mlib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarInit\\_F32\(3MLIB\)](#), [mlib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalar\\_F32\(3MLIB\)](#), [mlib\\_SignalDTWKScalarFree\\_S16\(3MLIB\)](#), [mlib\\_SignalDTWKScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWKScalar\_S16 – perform dynamic time warping for K-best paths on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKScalar_S16(mllib_d64 *dist,
 const mllib_s16 *dobs, mllib_s32 lobs, mllib_s32 sob, s,
 void *state);
```

**Description** The `mllib_SignalDTWKScalar_S16()` function performs dynamic time warping for K-best paths on scalar data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.



To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
----------*
|p4 |p1 |p0
| | |
----------*
| |p2 |
| | |
----------* px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive  $(1,0) (1,0)$  is disallowed. So path  $p4 \rightarrow p1 \rightarrow p0$  is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the  $(p_x, p_y)$  plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in  $(p_x, p_y)$  plane can be represented in chain code. The value of the chain code is defined as following.

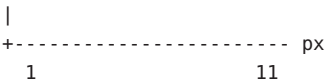
|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x- -0- -*- -*- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * _ *
|
| * * * * * * * _ * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * _ * * * * *
|
| * * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
1 | * * * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

- Parameters** The function takes the following arguments:
- dist* The distances of the K-best paths.
  - dobs* The observed data array.
  - lobs* The length of the observed data array.
  - sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2*(-scaling_factor)`.
  - state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKScalarFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWKVector\_F32 – perform dynamic time warping for K-best paths on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKVector_F32(mllib_d64 *dist,
 const mllib_f32 **dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWKVector_F32()` function performs dynamic time warping for K-best paths on vector data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{p_x(i), p_y(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(p_y(i)), o(p_x(i))) * m(p_x(i), p_y(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

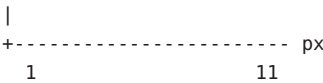
|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x- -0- -*- -*- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * * _ *
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
1 | * * * * * * * * * * /
```



The chain code that represents the path is  
(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

- Parameters** The function takes the following arguments:
- dist* The distances of the K-best paths.
  - dobs* The observed data array.
  - lobs* The length of the observed data array.
  - state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalDTWKVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKVectorPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKVectorFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWKVectorFree\_S16, mllib\_SignalDTWKVectorFree\_F32 – clean up for K-best paths of vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_SignalDTWKVectorFree_S16(void *state);
void mllib_SignalDTWKVectorFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for dynamic time warping (DTW) for K-best paths of vector data.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorPath\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalDTWKVectorInit\_F32 – initialization for K-best paths of vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalDTWKVectorInit_F32(void *state,
 const mllib_f32 **dref, mllib_s32 lref, mllib_s32 ndata,
 mllib_s32 kbest, mllib_s32 dtype, mllib_s32 delta,
 mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWKVectorInit_F32()` function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

|              |                                                                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dref</i>  | The reference data array.                                                                                                                                                                                           |
| <i>lref</i>  | The length of the reference data array.                                                                                                                                                                             |
| <i>ndata</i> | The length of each data vector.                                                                                                                                                                                     |
| <i>kbest</i> | The number of the best paths evaluated.                                                                                                                                                                             |
| <i>dtype</i> | The type of distance metric between data vectors. <code>MLIB_DTW_L1NORM</code> for L1 norm of difference (sum of absolute difference). <code>MLIB_DTW_L2NORM</code> for L2 norm of difference (Euclidean distance). |
| <i>delta</i> | The delta in the endpoint constraints.                                                                                                                                                                              |
| <i>local</i> | The type of the local continuity constraints. <code>MLIB_DTW_ITAKURA</code> for Itakura type constraints.                                                                                                           |
| <i>slope</i> | The type of the slope weighting. <code>MLIB_DTW_NONE</code> for no slope weighting.                                                                                                                                 |
| <i>state</i> | Pointer to the internal state structure.                                                                                                                                                                            |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWKVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_F32\(3MLIB\)](#),  
[mllib\\_SignalDTWKVectorPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKVectorFree\\_F32\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalDTWKVectorInit\_S16 – initialization for K-best paths of vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalDTWKVectorInit_S16(void *state,
 const mllib_s16 **dref, mllib_s32 lref, mllib_s32 ndata,
 mllib_s32 kbest, mllib_s32 dtype, mllib_s32 sref,
 mllib_s32 delta, mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWKVectorInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

|              |                                                                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dref</i>  | The reference data array.                                                                                                                                                                                           |
| <i>lref</i>  | The length of the reference data array.                                                                                                                                                                             |
| <i>ndata</i> | The length of each data vector.                                                                                                                                                                                     |
| <i>kbest</i> | The number of the best paths evaluated.                                                                                                                                                                             |
| <i>dtype</i> | The type of distance metric between data vectors. <code>MLIB_DTW_L1NORM</code> for L1 norm of difference (sum of absolute difference). <code>MLIB_DTW_L2NORM</code> for L2 norm of difference (Euclidean distance). |
| <i>sref</i>  | The scaling factor of the reference data array, where <code>actual_data = input_data * 2*(-scaling_factor)</code> .                                                                                                 |
| <i>delta</i> | The delta in the endpoint constraints.                                                                                                                                                                              |
| <i>local</i> | The type of the local continuity constraints. <code>MLIB_DTW_ITAKURA</code> for Itakura type constraints.                                                                                                           |
| <i>slope</i> | The type of the slope weighting. <code>MLIB_DTW_NONE</code> for no slope weighting.                                                                                                                                 |
| <i>state</i> | Pointer to the internal state structure.                                                                                                                                                                            |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mllib\\_SignalDTWKVectorInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_S16\(3MLIB\)](#),  
[mllib\\_SignalDTWKVectorPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorFree\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalDTWKVectorPath\_S16, mllib\_SignalDTWKVectorPath\_F32 – return K-best path on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKVectorPath_S16(mllib_s32 *path,
 mllib_s32 *lpath, mllib_s32 kpath, void *state);

mllib_status mllib_SignalDTWKVectorPath_F32(mllib_s32 *path,
 mllib_s32 *lpath, mllib_s32 kpath, void *state);
```

**Description** Each of these functions returns K-best path on vector data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{(r - o)**2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
----------*
|p4 |p1 |p0
| | |
----------*
| |p2 |
| | |
----------* px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive  $(1,0) (1,0)$  is disallowed. So path  $p4 \rightarrow p1 \rightarrow p0$  is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the  $(px, py)$  plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in  $(px, py)$  plane can be represented in chain code. The value of the chain code is defined as following.

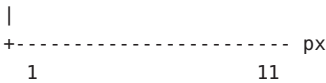
|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * *
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
1 | * * * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** Each of the functions takes the following arguments:

- path*      The optimal path.
- lpath*     The length of the optimal path.
- kpath*     The path index,  $0 \leq kpath < kbest$ .
- state*     Pointer to the internal state structure.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalDTWKScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWKScalarFree\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalDTWKVector\_S16 – perform dynamic time warping for K-best paths on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWKVector_S16(mllib_d64 *dist,
 const mllib_s16 **dobs, mllib_s32 lobs, mllib_s32 sob, s,
 void *state);
```

**Description** The `mllib_SignalDTWKVector_S16()` function performs dynamic time warping for K-best paths on vector data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

### 1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$p_x(Q) = M$$
$$N - \delta \leq p_y(Q) \leq N$$

## 2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
----------*
|p4 |p1 |p0
| | |
----------*
| |p2 |
| | |
----------* p3

```

Allowable paths are

|        |       |
|--------|-------|
| p1->p0 | (1,0) |
| p2->p0 | (1,1) |
| p3->p0 | (1,2) |

Consecutive  $(1, 0) (1, 0)$  is disallowed. So path  $p_4 \rightarrow p_1 \rightarrow p_0$  is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the  $(px, py)$  plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

## 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

```
=====
shift (x , y) | chain code

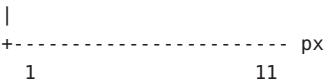
(1 , 0) | 0
(0 , 1) | 1
(1 , 1) | 2
(2 , 1) | 3
(1 , 2) | 4
(3 , 1) | 5
(3 , 2) | 6
(1 , 3) | 7
(2 , 3) | 8
=====
```

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * *
|
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
1 | * * * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distances of the K-best paths.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.
- state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWKVectorInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVector\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWKVectorFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalar\_F32 – perform dynamic time warping on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWScalar_F32(mllib_d64 *dist,
 const mllib_f32 *dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWScalar_F32()` function performs dynamic time warping on scalar data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r, o) = |r - o| = \text{SQRT} \{(r - o)^2\}$$

The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

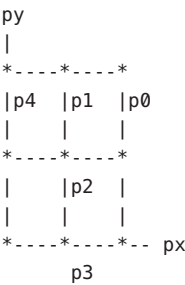
2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:



Allowable paths are

|        |       |
|--------|-------|
| p1->p0 | (1,0) |
| p2->p0 | (1,1) |
| p3->p0 | (1,2) |

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

```
=====
shift (x , y) | chain code

```

=====

The chain code that represents the path is

( 2 2 2 1 2 0 2 2 0 2 0 )

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distance of the optimal path.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalDTWScalarFree\_S16, mllib\_SignalDTWScalarFree\_F32 – clean up for scalar data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalDTWScalarFree_S16(void *state);
void mllib_SignalDTWScalarFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for dynamic time warping (DTW) of scalar data.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*     Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalarInit\_F32 – initialization for scalar data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalDTWScalarInit_F32(void *state,
 const mllib_f32 *dref, mllib_s32 lref, mllib_s32 delta,
 mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWScalarInit_F32()` function initializes the internal state structure for dynamic time warping (DTW) of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

*dref*      The reference data array.

*lref*      The length of the reference data array.

*delta*     The delta in the endpoint constraints.

*local*     The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.

*slope*     The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.

*state*     Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalarInit\_S16 – initialization for scalar data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalDTWScalarInit_S16(void *state,
 const mllib_s16 *dref, mllib_s32 lref, mllib_s32 sref,
 mllib_s32 delta, mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWScalarInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

*dref*      The reference data array.

*lref*      The length of the reference data array.

*sref*      The scaling factor of the reference data array, where `actual_data = input_data * 2**(-scaling_factor)`.

*delta*     The delta in the endpoint constraints.

*local*    The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.

*slope*    The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.

*state*    Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalarPath\_F32 – perform dynamic time warping on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWScalarPath_F32(mllib_d64 *dist, mllib_s32 *path,
 mllib_s32 *lpath, const mllib_f32 *dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWScalarPath_F32()` function performs dynamic time warping on scalar data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{p_x(i), p_y(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(p_y(i)), o(p_x(i))) * m(p_x(i), p_y(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

```
=====
shift (x , y) | chain code

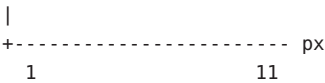
(1 , 0) | 0
(0 , 1) | 1
(1 , 1) | 2
(2 , 1) | 3
(1 , 2) | 4
(3 , 1) | 5
(3 , 2) | 6
(1 , 3) | 7
(2 , 3) | 8
=====
```

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * *
|
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * * /
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
| * * * * * * * * * *
1 | * * * * * * * * * *
```



The chain code that represents the path is  
(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distance of the optimal path.
- path* The optimal path.
- lpath* The length of the optimal path.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalDTWScalarInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalarPath\_S16 – perform dynamic time warping on scalar data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWScalarPath_S16(mllib_d64 *dist, mllib_s32 *path,
 mllib_s32 *lpath, const mllib_s16 *dobs, mllib_s32 lobs, mllib_s32 sob, s,
 void *state);
```

**Description** The `mllib_SignalDTWScalarPath_S16()` function performs dynamic time warping on scalar data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.



$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

```
=====
shift (x , y) | chain code

(1 , 0) | 0
(0 , 1) | 1
(1 , 1) | 2
(2 , 1) | 3
(1 , 2) | 4
(3 , 1) | 5
(3 , 2) | 6
(1 , 3) | 7
(2 , 3) | 8
=====
```

```

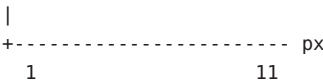
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px

```

where  $x$  marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
9 | * * * * * _ *
 | /
 | * * * * * _ * *
 | /
 | * * * * * * * *
 | /
 | * * * * * _ * * * *
 | /
 | * * * * * * * *
 | |
 | * * * * * * * *
 | /
 | * * * * * * * *
 | /
 | * * * * * * * *
 | /
 | * * * * * * * *
1 | * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

- Parameters** The function takes the following arguments:
- dist* The distance of the optimal path.
  - path* The optimal path.
  - lpath* The length of the optimal path.
  - dobs* The observed data array.
  - lobs* The length of the observed data array.
  - sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.
  - state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWScalar\_S16 – perform dynamic time warping on scalar data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalDTWScalar_S16(mllib_d64 *dist,
 const mllib_s16 *dobs, mllib_s32 lobs, mllib_s32 sob,
 void *state);
```

**Description** The `mllib_SignalDTWScalar_S16()` function performs dynamic time warping on scalar data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{p_x(i), p_y(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(p_y(i)), o(p_x(i))) * m(p_x(i), p_y(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

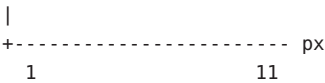
|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x- -0- -*- -*- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * * _ *
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
1 | * * * * * * * * * * *
```



The chain code that represents the path is  
(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

- Parameters** The function takes the following arguments:
- dist* The distance of the optimal path.
  - dobs* The observed data array.
  - lobs* The length of the observed data array.
  - sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2*(-scaling_factor)`.
  - state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWScalarInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalar\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWScalarFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVector\_F32 – perform dynamic time warping on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalDTWVector_F32(mllib_d64 *dist,
const mllib_f32 **dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWVector_F32()` function performs dynamic time warping on vector data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r, o) = |r - o| = \text{SQRT} \{(r - o)^2\}$$



The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

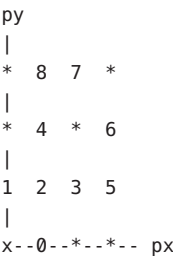
```

=====
shift (x , y) | chain code

```

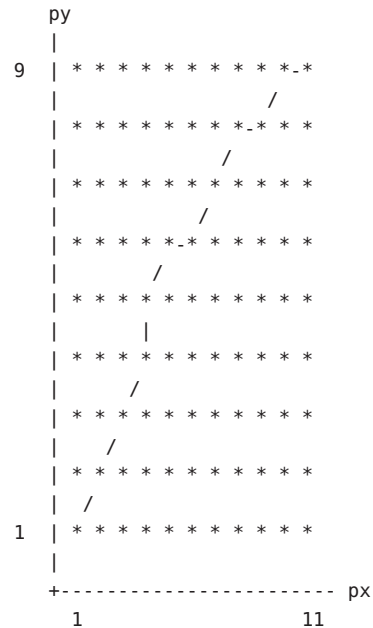
|           |  |   |
|-----------|--|---|
| ( 1 , 0 ) |  | 0 |
| ( 0 , 1 ) |  | 1 |
| ( 1 , 1 ) |  | 2 |
| ( 2 , 1 ) |  | 3 |
| ( 1 , 2 ) |  | 4 |
| ( 3 , 1 ) |  | 5 |
| ( 3 , 2 ) |  | 6 |
| ( 1 , 3 ) |  | 7 |
| ( 2 , 3 ) |  | 8 |

=====



where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points



The chain code that represents the path is

( 2 2 2 1 2 0 2 2 0 2 0 )

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distance of the optimal path.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVectorFree\_S16, mllib\_SignalDTWVectorFree\_F32 – clean up for vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalDTWVectorFree_S16(void *state);
void mllib_SignalDTWVectorFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for dynamic time warping (DTW) of vector data.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVectorInit\_F32 – initialization for vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalDTWVectorInit_F32(void *state,
 const mllib_f32 **dref, mllib_s32 lref, mllib_s32 ndata,
 mllib_s32 dtype, mllib_s32 delta, mllib_s32 local,
 mllib_s32 slope);
```

**Description** The `mllib_SignalDTWVectorInit_F32()` function initializes the internal state structure for dynamic time warping (DTW) of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

|              |                                                                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dref</i>  | The reference data array.                                                                                                                                                                                           |
| <i>lref</i>  | The length of the reference data array.                                                                                                                                                                             |
| <i>ndata</i> | The length of each data vector.                                                                                                                                                                                     |
| <i>dtype</i> | The type of distance metric between data vectors. <code>MLIB_DTW_L1NORM</code> for L1 norm of difference (sum of absolute difference). <code>MLIB_DTW_L2NORM</code> for L2 norm of difference (Euclidean distance). |
| <i>delta</i> | The delta in the endpoint constraints.                                                                                                                                                                              |
| <i>local</i> | The type of the local continuity constraints. <code>MLIB_DTW_ITAKURA</code> for Itakura type constraints.                                                                                                           |
| <i>slope</i> | The type of the slope weighting. <code>MLIB_DTW_NONE</code> for no slope weighting.                                                                                                                                 |
| <i>state</i> | Pointer to the internal state structure.                                                                                                                                                                            |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVectorInit\_S16 – initialization for vector data

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalDTWVectorInit_S16(void *state,
 const mllib_s16 **dref, mllib_s32 lref, mllib_s32 ndata,
 mllib_s32 dtype, mllib_s32 sref, mllib_s32 delta,
 mllib_s32 local, mllib_s32 slope);
```

**Description** The `mllib_SignalDTWVectorInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters** The function takes the following arguments:

*dref*      The reference data array.

*lref*      The length of the reference data array.

*ndata*     The length of each data vector.

*dtype*     The type of distance metric between data vectors. `MLIB_DTW_L1NORM` for L1 norm of difference (sum of absolute difference). `MLIB_DTW_L2NORM` for L2 norm of difference (Euclidean distance).

*sref*      The scaling factor of the reference data array, where `actual_data = input_data * 2*(-scaling_factor)`.

*delta*     The delta in the endpoint constraints.

*local*     The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.

*slope*     The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.

*state*     Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalDTWVectorInit_S16(3MLIB)`, `mlib_SignalDTWVector_S16(3MLIB)`,  
`mlib_SignalDTWVectorPath_S16(3MLIB)`, `mlib_SignalDTWVectorFree_S16(3MLIB)`,  
`attributes(5)`

**Name** mllib\_SignalDTWVectorPath\_F32 – perform dynamic time warping on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWVectorPath_F32(mllib_d64 *dist, mllib_s32 *path,
 mllib_s32 *lpath, const mllib_f32 **dobs, mllib_s32 lobs, void *state);
```

**Description** The `mllib_SignalDTWVectorPath_F32()` function performs dynamic time warping on vector data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{p_x(i), p_y(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(p_y(i)), o(p_x(i))) * m(p_x(i), p_y(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.



$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

#### 1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

#### 2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

#### 3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*--- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

#### 4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

#### 5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

```
=====
shift (x , y) | chain code

(1 , 0) | 0
(0 , 1) | 1
(1 , 1) | 2
(2 , 1) | 3
(1 , 2) | 4
(3 , 1) | 5
(3 , 2) | 6
(1 , 3) | 7
(2 , 3) | 8
=====
```

```

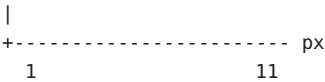
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px

```

where  $x$  marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
9 | * * * * * _ *
 | * * * * * /
 | * * * * * _ *
 | * * * * * /
 | * * * * * _ *
 | * * * * * /
 | * * * * * |
 | * * * * * /
 | * * * * * /
 | * * * * * /
1 | * * * * *
```



The chain code that represents the path is  
( 2 2 2 1 2 0 2 2 0 2 0 )

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- dist* The distance of the optimal path.
- path* The optimal path.
- lpath* The length of the optimal path.
- dobs* The observed data array.
- lobs* The length of the observed data array.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_F32\(3MLIB\)](#), [mllib\\_SignalDTWVectorFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVectorPath\_S16 – perform dynamic time warping on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalDTWVectorPath_S16(mllib_d64 *dist, mllib_s32 *path,
 mllib_s32 *lpath, const mllib_s16 **dobs, mllib_s32 lobs, mllib_s32 sob, s,
 void *state);
```

**Description** The `mllib_SignalDTWVectorPath_S16()` function performs dynamic time warping on vector data.

Assume the reference data are

$$r(y), \quad y=1, 2, \dots, N$$

and the observed data are

$$o(x), \quad x=1, 2, \dots, M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1, 2, \dots, Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \sqrt{\sum_{i=0}^{L-1} (r(i) - o(i))^2}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.

$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} px(1) &= 1 \\ 1 \leq py(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} px(Q) &= M \\ N - \text{delta} \leq py(Q) &\leq N \end{aligned}$$

2. Monotonicity Conditions

$$\begin{aligned} px(i) &\leq px(i+1) \\ py(i) &\leq py(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

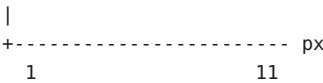
|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x- -0- -*- -*- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * * _ *
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
| * * * * * * * * * * /
|
1 | * * * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

- Parameters** The function takes the following arguments:
- dist* The distance of the optimal path.
  - path* The optimal path.
  - lpath* The length of the optimal path.
  - dobs* The observed data array.
  - lobs* The length of the observed data array.
  - sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.
  - state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalDTWVector\_S16 – perform dynamic time warping on vector data

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalDTWVector_S16(mllib_d64 *dist,
 const mllib_s16 **dobs, mllib_s32 lobs, mllib_s32 sob,
 void *state);
```

**Description** The `mllib_SignalDTWVector_S16()` function performs dynamic time warping on vector data.

Assume the reference data are

$$r(y), \quad y=1,2,\dots,N$$

and the observed data are

$$o(x), \quad x=1,2,\dots,M$$

the dynamic time warping is to find a mapping function (a path)

$$p(i) = \{px(i), py(i)\}, \quad i=1,2,\dots,Q$$

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

$$\text{dist} = \sum_{i=1}^Q d(r(py(i)), o(px(i))) * m(px(i), py(i))$$

where  $d(r, o)$  is the dissimilarity between data point/vector  $r$  and data point/vector  $o$ ;  $m(x, y)$  is the path weighting coefficient associated with path point  $(x, y)$ ;  $N$  is the length of the reference data;  $M$  is the length of the observed data;  $Q$  is the length of the path.

Using L1 norm (sum of absolute differences)

$$d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|$$

Using L2 norm (Euclidean distance)

$$d(r, o) = \text{SQRT} \left\{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 \right\}$$

where  $L$  is the length of each data vector.

To scalar data where  $L=1$ , the two norms are the same.



$$d(r,o) = |r - o| = \text{SQRT} \{ (r - o)^2 \}$$

The constraints of dynamic time warping are:

1. Endpoint constraints

$$\begin{aligned} p_x(1) &= 1 \\ 1 \leq p_y(1) &\leq 1 + \text{delta} \end{aligned}$$

and

$$\begin{aligned} p_x(Q) &= M \\ N - \text{delta} \leq p_y(Q) &\leq N \end{aligned}$$

2. Monotonicity Conditions

$$\begin{aligned} p_x(i) &\leq p_x(i+1) \\ p_y(i) &\leq p_y(i+1) \end{aligned}$$

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang's book.

Itakura Type:

```

py
|
------*
|p4 |p1 |p0
| | |
------*
| |p2 |
| | |
------*-- px
 p3

```

Allowable paths are

```

p1->p0 (1,0)
p2->p0 (1,1)
p3->p0 (1,2)

```

Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.

|                              |  |   |
|------------------------------|--|---|
| =====                        |  |   |
| shift ( x , y )   chain code |  |   |
| -----                        |  |   |
| ( 1 , 0 )                    |  | 0 |
| ( 0 , 1 )                    |  | 1 |
| ( 1 , 1 )                    |  | 2 |
| ( 2 , 1 )                    |  | 3 |
| ( 1 , 2 )                    |  | 4 |
| ( 3 , 1 )                    |  | 5 |
| ( 3 , 2 )                    |  | 6 |
| ( 1 , 3 )                    |  | 7 |
| ( 2 , 3 )                    |  | 8 |
| =====                        |  |   |

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
9 | * * * * * * * * * _ *
|
| * * * * * * * _ * * *
|
| * * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * _ * * * * *
|
| * * * * * * * * * *
|
| * * * * * * * * * *
|
| * * * * * * * * * *
|
| * * * * * * * * *
|
| * * * * * * * * *
|
| /
|
1 | * * * * * * * * * *
```



The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*dist* The distance of the optimal path.

*dobs* The observed data array.

*lobs* The length of the observed data array.

*sobs* The scaling factor of the observed data array, where `actual_data = input_data * 2*(-scaling_factor)`.

*state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDTWVectorInit\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVector\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorPath\\_S16\(3MLIB\)](#), [mllib\\_SignalDTWVectorFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalEmphasizeFree\_S16\_S16, mllib\_SignalEmphasizeFree\_S16S\_S16S, mllib\_SignalEmphasizeFree\_F32\_F32, mllib\_SignalEmphasizeFree\_F32S\_F32S – clean up for signal pre-emphasizing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalEmphasizeFree_S16_S16(void *filter);
void mllib_SignalEmphasizeFree_S16S_S16S(void *filter);
void mllib_SignalEmphasizeFree_F32_F32(void *filter);
void mllib_SignalEmphasizeFree_F32S_F32S(void *filter);
```

**Description** Each of these functions releases the memory allocated for the internal state's structure.

**Parameters** Each of the functions takes the following arguments:  
*filter*      Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalEmphasize\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalEmphasizeInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalEmphasizeInit\_S16\_S16, mllib\_SignalEmphasizeInit\_S16S\_S16S, mllib\_SignalEmphasizeInit\_F32\_F32, mllib\_SignalEmphasizeInit\_F32S\_F32S – initialization for signal pre-emphasizing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalEmphasizeInit_S16_S16(void **filter,
 mllib_f32 alpha);

mllib_status mllib_SignalEmphasizeInit_S16S_S16S(void **filter,
 mllib_f32 alpha);

mllib_status mllib_SignalEmphasizeInit_F32_F32(void **filter,
 mllib_f32 alpha);

mllib_status mllib_SignalEmphasizeInit_F32S_F32S(void **filter,
 mllib_f32 alpha);
```

**Description** Each of these functions allocates memory for an internal filter structure and converts the filter coefficients into the internal representation.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

*alpha* Emphasizing coefficient.  $0 < \alpha < 1.0$

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalEmphasize\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalEmphasizeFree\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalEmphasize\_S16\_S16\_Sat, mllib\_SignalEmphasize\_S16S\_S16S\_Sat, mllib\_SignalEmphasize\_F32\_F32, mllib\_SignalEmphasize\_F32S\_F32S – signal pre-emphasizing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalEmphasize_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalEmphasize_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalEmphasize_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalEmphasize_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);
```

**Description** Each of these functions applies the preemphasizer to one signal packet and updates the filter states.

For monaural signals, the following equation is used:

$$dst[i] = src[i] - \alpha * src[i - 1]$$

where  $i = 0, 1, \dots, (n - 1)$ ;  $src[-1] = 0$ .

For stereo signals, the following equation is used:

$$\begin{aligned} dst[2*i] &= src[2*i] - \alpha * src[2*(i - 1)] \\ dst[2*i + 1] &= src[2*i + 1] - \alpha * src[2*(i - 1) + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ ;  $src[-2] = src[-1] = 0$ .

**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src* Source signal array.
- filter* Internal filter structure.
- n* Number of samples in the source signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mlib\\_SignalEmphasizeFree\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalEmphasizeInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFFT\_1, mllib\_SignalFFT\_1\_S16\_S16\_Mod,  
 mllib\_SignalFFT\_1\_S16C\_S16C\_Mod, mllib\_SignalFFT\_1\_S16C\_S16\_Mod,  
 mllib\_SignalFFT\_1\_S16\_Mod, mllib\_SignalFFT\_1\_S16C\_Mod, mllib\_SignalFFT\_1\_F32\_F32,  
 mllib\_SignalFFT\_1\_F32C\_F32C, mllib\_SignalFFT\_1\_F32C\_F32, mllib\_SignalFFT\_1\_F32,  
 mllib\_SignalFFT\_1\_F32C, mllib\_SignalFFT\_1\_D64\_D64, mllib\_SignalFFT\_1\_D64C\_D64C,  
 mllib\_SignalFFT\_1\_D64C\_D64, mllib\_SignalFFT\_1\_D64, mllib\_SignalFFT\_1\_D64C – signal  
 Fast Fourier Transform (FFT)

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
 #include <mllib.h>

```
mllib_status mllib_SignalFFT_1_S16_S16_Mod(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_S16C_S16C_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcc, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_S16C_S16_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcr, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_S16_Mod(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_S16C_Mod(mllib_s16 *srcdstc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_F32_F32(mllib_f32 *dstr,
 mllib_f32 *dsti, const mllib_f32 *srcr, const mllib_f32 *srci,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *srcc, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_F32C_F32(mllib_f32 *dstc,
 const mllib_f32 *srcr, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_F32(mllib_f32 *srcdstr,
 mllib_f32 *srcdsti, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_F32C(mllib_f32 *srcdstc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_D64_D64(mllib_d64 *dstr,
 mllib_d64 *dsti, const mllib_d64 *srcr, const mllib_d64 *srci,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_D64C_D64C(mllib_d64 *dstc,
 const mllib_d64 *srcc, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_D64C_D64(mllib_d64 *dstc,
 const mllib_d64 *srcr, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_1_D64(mllib_d64 *srcdstr,
 mllib_d64 *srcdsti, mllib_s32 order);
```



```
mllib_status mllib_SignalFFT_1_D64C(mllib_d64 *srcdstc,
 mllib_s32 order);
```

**Description** Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

$$\text{dst}[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \exp(j2\pi k*n/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}\text{order}$

The signal FFT/IFFT functions can be categorized into four groups according to the `ScaleMode` in the function names in the following form:

```
mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()
```

The scaling factors `C1` and `C2` used in the equations are defined as follows:

- For `ScaleMode = 1`, `C1 = 1` and `C2 = 2**order`.
- For `ScaleMode = 2`, `C1 = 2**order` and `C2 = 1`.
- For `ScaleMode = 3`, `C1 = C2 = 2**((order/2))` when `order` is even, or `C1 = 2**((order+1)/2)` and `C2 = 2**((order-1)/2)` when `order` is odd.
- For `ScaleMode = 4`, `C1 = 2**P` and `C2 = 2**Q`, where `P` and `Q` are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

*dst* Destination signal array that contains the real parts.  
*dsti* Destination signal array that contains the imaginary parts.  
*srcr* Source signal array that contains the real parts.  
*srci* Source signal array that contains the imaginary parts.

- dstc*      Complex destination signal array. *dstc*[2\*i] contains the real parts, and *dstc*[2\*i+1] contains the imaginary parts.
- srcc*      Complex source signal array. *srcc*[2\*i] contains the real parts, and *srcc*[2\*i+1] contains the imaginary parts.
- srcdstr*    Source and destination signal array that contains the real parts.
- srcdsti*    Source and destination signal array that contains the imaginary parts.
- srcdstc*    Complex source and destination signal array. *srcdstc*[2\*i] contains the real parts, and *srcdstc*[2\*i+1] contains the imaginary parts.
- order*      Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#),  
[mllib\\_SignalIFFT\\_1\(3MLIB\)](#), [mllib\\_SignalIFFT\\_2\(3MLIB\)](#), [mllib\\_SignalIFFT\\_3\(3MLIB\)](#),  
[mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalFFT\_2, mlib\_SignalFFT\_2\_S16\_S16, mlib\_SignalFFT\_2\_S16C\_S16C, mlib\_SignalFFT\_2\_S16C\_S16, mlib\_SignalFFT\_2\_S16, mlib\_SignalFFT\_2\_S16C, mlib\_SignalFFT\_2\_F32\_F32, mlib\_SignalFFT\_2\_F32C\_F32C, mlib\_SignalFFT\_2\_F32C\_F32, mlib\_SignalFFT\_2\_F32, mlib\_SignalFFT\_2\_F32C, mlib\_SignalFFT\_2\_D64\_D64, mlib\_SignalFFT\_2\_D64C\_D64C, mlib\_SignalFFT\_2\_D64C\_D64, mlib\_SignalFFT\_2\_D64, mlib\_SignalFFT\_2\_D64C – signal Fast Fourier Transform (FFT)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```

mlib_status mlib_SignalFFT_2_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti,
 const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C_S16C(mlib_s16 *dstc,
 const mlib_s16 *srcc,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C_S16(mlib_s16 *dstc, const mlib_s16 *srcr,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16(mlib_s16 *srcdstr, mlib_s16 *srcdsti,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C(mlib_s16 *srcdstc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32_F32(mlib_f32 *dstr,
 mlib_f32 *dsti, const mlib_f32 *srcr,
 const mlib_f32 *srci,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C_F32C(mlib_f32 *dstc,
 const mlib_f32 *srcc,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C_F32(mlib_f32 *dstc, const mlib_f32 *srcr,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32(mlib_f32 *srcdstr, mlib_f32 *srcdsti,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C(mlib_f32 *srcdstc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64_D64(mlib_d64 *dstr, mlib_d64 *dsti,
 const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64C_D64C(mlib_d64 *dstc,
 const mlib_d64 *srcc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64C_D64(mlib_d64 *dstc, const mlib_d64 *srcr,
 mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64(mlib_d64 *srcdstr, mlib_d64 *srcdsti,
 mlib_s32 order);

```

```
mllib_status mllib_SignalFFT_2_D64C(mllib_d64 *srcdstc, mllib_s32 order);
```

**Description** Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

$$\text{dst}[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$

$n = 0, 1, \dots, (N - 1)$

$N = 2^{**}\text{order}$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\* (order/2) when order is even, or C1 = 2\*\* ((order+1)/2) and C2 = 2\*\* ((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

*dst* Destination signal array that contains the real parts.

*dsti* Destination signal array that contains the imaginary parts.

*src* Source signal array that contains the real parts.

*srci* Source signal array that contains the imaginary parts.

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                  |
| <i>srcc</i>    | Complex source signal array. <code>srcc[2*i]</code> contains the real parts, and <code>srcc[2*i+1]</code> contains the imaginary parts.                       |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                             |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                        |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                              |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFT\\_1\(3MLIB\)](#), [mllib\\_SignalIFFT\\_2\(3MLIB\)](#), [mllib\\_SignalIFFT\\_3\(3MLIB\)](#), [mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFFT\_3, mllib\_SignalFFT\_3\_S16\_S16\_Mod,  
mllib\_SignalFFT\_3\_S16C\_S16C\_Mod, mllib\_SignalFFT\_3\_S16C\_S16\_Mod,  
mllib\_SignalFFT\_3\_S16\_Mod, mllib\_SignalFFT\_3\_S16C\_Mod, mllib\_SignalFFT\_3\_F32\_F32,  
mllib\_SignalFFT\_3\_F32C\_F32C, mllib\_SignalFFT\_3\_F32C\_F32, mllib\_SignalFFT\_3\_F32,  
mllib\_SignalFFT\_3\_F32C, mllib\_SignalFFT\_3\_D64\_D64, mllib\_SignalFFT\_3\_D64C\_D64C,  
mllib\_SignalFFT\_3\_D64C\_D64, mllib\_SignalFFT\_3\_D64, mllib\_SignalFFT\_3\_D64C – signal  
Fast Fourier Transform (FFT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalFFT_3_S16_S16_Mod(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_S16C_S16C_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_S16C_S16_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcr,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_S16_Mod(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_S16C_Mod(mllib_s16 *srcdstc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_F32_F32(mllib_f32 *dstr,
 mllib_f32 *dsti,
 const mllib_f32 *srcr, const mllib_f32 *srci, mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *srcc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_F32C_F32(mllib_f32 *dstc,
 const mllib_f32 *srcr,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_F32(mllib_f32 *srcdstr,
 mllib_f32 *srcdsti,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_F32C(mllib_f32 *srcdstc,
 mllib_s32 order);
```

```
mllib_status mllib_SignalFFT_3_D64_D64(mllib_d64 *dstr,
 mllib_d64 *dsti,
 const mllib_d64 *srcr, const mllib_d64 *srci, mllib_s32 order);
```

```

mllib_status mllib_SignalFFT_3_D64C_D64C(mllib_d64 *dstc,
 const mllib_d64 *src,
 mllib_s32 order);

mllib_status mllib_SignalFFT_3_D64C_D64(mllib_d64 *dstc,
 const mllib_d64 *src,
 mllib_s32 order);

mllib_status mllib_SignalFFT_3_D64(mllib_d64 *srcdst,
 mllib_d64 *srcdsti,
 mllib_s32 order);

mllib_status mllib_SignalFFT_3_D64C(mllib_d64 *srcdstc,
 mllib_s32 order);

```

**Description** Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

$$dst[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{src[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$dst[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{src[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}order$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```

mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- dstr* Destination signal array that contains the real parts.
- dsti* Destination signal array that contains the imaginary parts.
- srcr* Source signal array that contains the real parts.
- srci* Source signal array that contains the imaginary parts.
- dstc* Complex destination signal array. *dstc[2\*i]* contains the real parts, and *dstc[2\*i+1]* contains the imaginary parts.
- srcc* Complex source signal array. *srcc[2\*i]* contains the real parts, and *srcc[2\*i+1]* contains the imaginary parts.
- srcdstr* Source and destination signal array that contains the real parts.
- srcdsti* Source and destination signal array that contains the imaginary parts.
- srcdstc* Complex source and destination signal array. *srcdstc[2\*i]* contains the real parts, and *srcdstc[2\*i+1]* contains the imaginary parts.
- order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFT\\_1\(3MLIB\)](#), [mllib\\_SignalIFFT\\_2\(3MLIB\)](#), [mllib\\_SignalIFFT\\_3\(3MLIB\)](#), [mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalFFT\_4, mllib\_SignalFFT\_4\_S16\_S16, mllib\_SignalFFT\_4\_S16C\_S16C, mllib\_SignalFFT\_4\_S16C\_S16, mllib\_SignalFFT\_4\_S16, mllib\_SignalFFT\_4\_S16C – signal Fast Fourier Transform (FFT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalFFT_4_S16_S16(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, mllib_s32 order,
 mllib_s32 *scale);

mllib_status mllib_SignalFFT_4_S16C_S16C(mllib_s16 *dstc,
 const mllib_s16 *srcr,
 mllib_s32 order, mllib_s32 *scale);

mllib_status mllib_SignalFFT_4_S16C_S16(mllib_s16 *dstc,
 const mllib_s16 *srcr,
 mllib_s32 order, mllib_s32 *scale);

mllib_status mllib_SignalFFT_4_S16(mllib_s16 *srcdstr, mllib_s16 *srcdsti,
 mllib_s32 order, mllib_s32 *scale);

mllib_status mllib_SignalFFT_4_S16C(mllib_s16 *srcdstc, mllib_s32 order,
 mllib_s32 *scale);
```

**Description** Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

$$\text{dst}[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}\text{order}$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.

- For `ScaleMode = 2`,  $C1 = 2^{**order}$  and  $C2 = 1$ .
- For `ScaleMode = 3`,  $C1 = C2 = 2^{** (order/2)}$  when `order` is even, or  $C1 = 2^{** ((order+1)/2)}$  and  $C2 = 2^{** ((order-1)/2)}$  when `order` is odd.
- For `ScaleMode = 4`,  $C1 = 2^{**P}$  and  $C2 = 2^{**Q}$ , where `P` and `Q` are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- dst* Destination signal array that contains the real parts.
- dsti* Destination signal array that contains the imaginary parts.
- src* Source signal array that contains the real parts.
- srci* Source signal array that contains the imaginary parts.
- dstc* Complex destination signal array. `dstc[2*i]` contains the real parts, and `dstc[2*i+1]` contains the imaginary parts.
- src* Complex source signal array. `src[2*i]` contains the real parts, and `src[2*i+1]` contains the imaginary parts.
- srcdst* Source and destination signal array that contains the real parts.
- srcdsti* Source and destination signal array that contains the imaginary parts.
- srcdstc* Complex source and destination signal array. `srcdstc[2*i]` contains the real parts, and `srcdstc[2*i+1]` contains the imaginary parts.
- order* Order of the transformation. The base-2 logarithm of the number of data samples.
- scale* Adaptive scaling factor.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalFFT_1(3MLIB)`, `mlib_SignalFFT_2(3MLIB)`, `mlib_SignalFFT_3(3MLIB)`,  
`mlib_SignalIFFT_1(3MLIB)`, `mlib_SignalIFFT_2(3MLIB)`, `mlib_SignalIFFT_3(3MLIB)`,  
`mlib_SignalIFFT_4(3MLIB)`, `attributes(5)`

**Name** mllib\_SignalFFTW\_1, mllib\_SignalFFTW\_1\_S16\_S16\_Mod, mllib\_SignalFFTW\_1\_S16C\_S16C\_Mod, mllib\_SignalFFTW\_1\_S16C\_S16\_Mod, mllib\_SignalFFTW\_1\_S16\_Mod, mllib\_SignalFFTW\_1\_S16C\_Mod, mllib\_SignalFFTW\_1\_F32\_F32, mllib\_SignalFFTW\_1\_F32C\_F32C, mllib\_SignalFFTW\_1\_F32C\_F32, mllib\_SignalFFTW\_1\_F32, mllib\_SignalFFTW\_1\_F32C – signal Fast Fourier Transform with windowing (FFTW)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalFFTW_1_S16_S16_Mod(mllib_s16 *dst, mllib_s16 *dsti,
 const mllib_s16 *src, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order);

mllib_status mllib_SignalFFTW_1_S16C_S16C_Mod(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_S16C_S16_Mod(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_S16_Mod(mllib_s16 *srcdst,
 mllib_s16 *srcdsti, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_S16C_Mod(mllib_s16 *srcdst,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_F32_F32(mllib_f32 *dst,
 mllib_f32 *dsti, const mllib_f32 *src, const mllib_f32 *srci,
 const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_F32C_F32C(mllib_f32 *dst,
 const mllib_f32 *src, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_F32C_F32(mllib_f32 *dst,
 const mllib_f32 *src, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_F32(mllib_f32 *srcdst,
 mllib_f32 *srcdsti, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_1_F32C(mllib_f32 *srcdst,
 const mllib_f32 *window, mllib_s32 order);
```

**Description** Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$

$n = 0, 1, \dots, (N - 1)$

$N = 2^{**}\text{order}$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                              |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                         |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                   |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                              |
| <i>dstc</i>    | Complex destination signal array. <i>dstc</i> [2*i] contains the real parts, and <i>dstc</i> [2*i+1] contains the imaginary parts.                  |
| <i>srcc</i>    | Complex source signal array. <i>srcc</i> [2*i] contains the real parts, and <i>srcc</i> [2*i+1] contains the imaginary parts.                       |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                   |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                              |
| <i>srcdstc</i> | Complex source and destination signal array. <i>srcdstc</i> [2*i] contains the real parts, and <i>srcdstc</i> [2*i+1] contains the imaginary parts. |

*window*     Window coefficient array with  $2^{**}order$  real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

*order*     Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values**   Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**   See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**   [mllib\\_SignalFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalFFTW\\_4\(3MLIB\)](#),  
[mllib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_2\(3MLIB\)](#),  
[mllib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFFTW\_2, mllib\_SignalFFTW\_2\_S16\_S16, mllib\_SignalFFTW\_2\_S16C\_S16C, mllib\_SignalFFTW\_2\_S16C\_S16, mllib\_SignalFFTW\_2\_S16, mllib\_SignalFFTW\_2\_S16C, mllib\_SignalFFTW\_2\_F32\_F32, mllib\_SignalFFTW\_2\_F32C\_F32C, mllib\_SignalFFTW\_2\_F32C\_F32, mllib\_SignalFFTW\_2\_F32, mllib\_SignalFFTW\_2\_F32C – signal Fast Fourier Transform with windowing (FFTW)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalFFTW_2_S16_S16(mllib_s16 *dst, mllib_s16 *dsti,
 const mllib_s16 *src, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order);

mllib_status mllib_SignalFFTW_2_S16C_S16C(mllib_s16 *dstc,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_S16C_S16(mllib_s16 *dstc,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_S16(mllib_s16 *srcdst,
 mllib_s16 *srcdsti, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_S16C(mllib_s16 *srcdstc,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_F32_F32(mllib_f32 *dst, mllib_f32 *dsti,
 const mllib_f32 *src, const mllib_f32 *srci, const mllib_f32 *
 window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *src, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_F32C_F32(mllib_f32 *dstc,
 const mllib_f32 *src, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_F32(mllib_f32 *srcdst,
 mllib_f32 *srcdsti, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_2_F32C(mllib_f32 *srcdstc,
 const mllib_f32 *window, mllib_s32 order);
```

**Description** Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] * \text{window}[n] * \exp(-j2\pi n * k / N)$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}\text{order}$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                              |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                         |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                   |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                              |
| <i>dstc</i>    | Complex destination signal array. <i>dstc</i> [2*i] contains the real parts, and <i>dstc</i> [2*i+1] contains the imaginary parts.                  |
| <i>srcc</i>    | Complex source signal array. <i>srcc</i> [2*i] contains the real parts, and <i>srcc</i> [2*i+1] contains the imaginary parts.                       |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                   |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                              |
| <i>srcdstc</i> | Complex source and destination signal array. <i>srcdstc</i> [2*i] contains the real parts, and <i>srcdstc</i> [2*i+1] contains the imaginary parts. |



*window* Window coefficient array with  $2^{**}order$  real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

*order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalFFTW\\_1\(3MLIB\)](#), [mlib\\_SignalFFTW\\_3\(3MLIB\)](#), [mlib\\_SignalFFTW\\_4\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_2\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFFTW\_3, mllib\_SignalFFTW\_3\_S16\_S16\_Mod, mllib\_SignalFFTW\_3\_S16C\_S16C\_Mod, mllib\_SignalFFTW\_3\_S16C\_S16\_Mod, mllib\_SignalFFTW\_3\_S16\_Mod, mllib\_SignalFFTW\_3\_S16C\_Mod, mllib\_SignalFFTW\_3\_F32\_F32, mllib\_SignalFFTW\_3\_F32C\_F32C, mllib\_SignalFFTW\_3\_F32C\_F32, mllib\_SignalFFTW\_3\_F32, mllib\_SignalFFTW\_3\_F32C – signal Fast Fourier Transform with windowing (FFTW)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalFFTW_3_S16_S16_Mod(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order);

mllib_status mllib_SignalFFTW_3_S16C_S16C_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcr, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_S16C_S16_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcr, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_S16_Mod(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti, const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_S16C_Mod(mllib_s16 *srcdstc,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_F32_F32(mllib_f32 *dstr,
 mllib_f32 *dsti, const mllib_f32 *srcr, const mllib_f32 *srci,
 const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *srcr, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_F32C_F32(mllib_f32 *dstc,
 const mllib_f32 *srcr, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_F32(mllib_f32 *srcdstr,
 mllib_f32 *srcdsti, const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalFFTW_3_F32C(mllib_f32 *srcdstc,
 const mllib_f32 *window, mllib_s32 order);
```

**Description** Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$

$n = 0, 1, \dots, (N - 1)$

$N = 2^{**}\text{order}$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order+1)/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                              |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                         |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                   |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                              |
| <i>dstc</i>    | Complex destination signal array. <i>dstc</i> [2*i] contains the real parts, and <i>dstc</i> [2*i+1] contains the imaginary parts.                  |
| <i>srcc</i>    | Complex source signal array. <i>srcc</i> [2*i] contains the real parts, and <i>srcc</i> [2*i+1] contains the imaginary parts.                       |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                   |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                              |
| <i>srcdstc</i> | Complex source and destination signal array. <i>srcdstc</i> [2*i] contains the real parts, and <i>srcdstc</i> [2*i+1] contains the imaginary parts. |

*window* Window coefficient array with  $2^{**}order$  real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

*order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalFFTW\\_4\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFFTW\_4, mllib\_SignalFFTW\_4\_S16\_S16, mllib\_SignalFFTW\_4\_S16C\_S16C, mllib\_SignalFFTW\_4\_S16C\_S16, mllib\_SignalFFTW\_4\_S16, mllib\_SignalFFTW\_4\_S16C – signal Fast Fourier Transform with windowing (FFTW)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalFFTW_4_S16_S16(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order, mllib_s32 *scale);

mllib_status mllib_SignalFFTW_4_S16C_S16C(mllib_s16 *dstc,
 const mllib_s16 *srcr, const mllib_s16 *window, mllib_s32 order,
 mllib_s32 *scale);

mllib_status mllib_SignalFFTW_4_S16C_S16(mllib_s16 *dstc,
 const mllib_s16 *srcr, const mllib_s16 *window, mllib_s32 order,
 mllib_s32 *scale);

mllib_status mllib_SignalFFTW_4_S16(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti, const mllib_s16 *window, mllib_s32 order,
 mllib_s32 *scale);

mllib_status mllib_SignalFFTW_4_S16C(mllib_s16 *srcdstc,
 const mllib_s16 *window, mllib_s32 order, mllib_s32 *scale);
```

**Description** Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2 * \text{order}$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*(order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                                           |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                                      |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                                |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                                           |
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                     |
| <i>src</i>     | Complex source signal array. <code>src[2*i]</code> contains the real parts, and <code>src[2*i+1]</code> contains the imaginary parts.                            |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                                |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                           |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts.    |
| <i>window</i>  | Window coefficient array with 2**order real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                                 |
| <i>scale</i>   | Adaptive scaling factor.                                                                                                                                         |

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

---

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalFFTW\\_1\(3MLIB\)](#), [mlib\\_SignalFFTW\\_2\(3MLIB\)](#), [mlib\\_SignalFFTW\\_3\(3MLIB\)](#),  
[mlib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_2\(3MLIB\)](#),  
[mlib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mlib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFIR\_F32\_F32 – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalFIR_F32_F32(mllib_f32 *dst, const mllib_f32 *src,
void *filter, mllib_s32 n);
```

**Description** The `mllib_SignalFIR_F32_F32()` function applies the FIR filter to one signal packet and updates the filter state.

**Parameters** The function takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- filter*     Internal filter structure.
- n*          Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalFIR\_F32S\_F32S – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalFIR_F32S_F32S(mllib_f32 *dst, const mllib_f32 *src,
void *filter, mllib_s32 n);
```

**Description** The `mllib_SignalFIR_F32S_F32S()` function applies the FIR filter to one signal packet and updates the filter state.

**Parameters** The function takes the following arguments:

*dst*        Output stereo signal array .. `dst[2*i]` contains Channel 0, and `dst[2*i+1]` contains Channel 1.

*src*        Input stereo signal array. `src[2*i]` contains Channel 0, and `src[2*i+1]` contains Channel 1.

*filter*     Internal filter structure.

*n*         Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalFIRFree\_F32\_F32 – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalFIRFree_F32_F32(void *filter);
```

**Description** The `mllib_SignalFIRFree_F32_F32()` function releases the memory allocated for the internal filter structure.

**Parameters** The function takes the following arguments:  
  
*filter*      Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mlib\_SignalFIRFree\_F32S\_F32S – Finite Impulse Response (FIR) filtering

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

`void mlib_SignalFIRFree_F32S_F32S(void *filter);`

**Description** The `mlib_SignalFIRFree_F32S_F32S()` function releases the memory allocated for the internal filter structure.

**Parameters** The function takes the following arguments:

*filter*      Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalFIRFree\_S16\_S16, mllib\_SignalFIRFree\_S16S\_S16S – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalFIRFree_S16_S16(void *filter);
void mllib_SignalFIRFree_S16S_S16S(void *filter);
```

**Description** Each of these functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#), [mllib\\_SignalFIRInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalFIRInit\_F32\_F32 – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalFIRInit_F32_F32(void **filter,
 const mlib_f32 *flt, mlib_s32 tap);
```

**Description** The `mlib_SignalFIRInit_F32_F32()` function allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

**Parameters** The function takes the following arguments:

*filter* Internal filter structure.

*flt* Filter coefficient array.

*tap* Taps of the filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalFIRInit\_F32S\_F32S – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalFIRInit_F32S_F32S(void **filter,
 const mllib_f32 *flt, mllib_s32 tap);
```

**Description** The `mllib_SignalFIRInit_F32S_F32S()` function allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

**Parameters** The function takes the following arguments:

- filter* Internal filter structure.
- flt* Filter coefficient array in stereo format. `flt[2*i]` contains Channel 0, and `flt[2*i+1]` contains Channel 1
- tap* Taps of the filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mlib\_SignalFIRInit\_S16\_S16, mlib\_SignalFIRInit\_S16S\_S16S – Finite Impulse Response (FIR) filtering

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalFIRInit_S16_S16(void **filter,
 const mlib_f32 *flt, mlib_s32 tap);

mlib_status mlib_SignalFIRInit_S16S_S16S(void **filter,
 const mlib_f32 *flt, mlib_s32 tap);
```

**Description** Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

**Parameters** Each of the functions takes the following arguments:

*filter*     Internal filter structure.  
*flt*        Filter coefficient array.  
*tap*        Taps of the filter.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#), [mlib\\_SignalFIRFree\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalFIR\_S16\_S16\_Sat, mllib\_SignalFIR\_S16S\_S16S\_Sat – Finite Impulse Response (FIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalFIR_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalFIR_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);
```

**Description** Each of these functions applies the FIR filter to one signal packet and updates the filter state.

**Parameters** Each of the functions takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- filter*     Internal filter structure.
- n*          Number of samples in the input signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFIRFree\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalFIRInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalGaussNoise\_F32 – Gaussian noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalGaussNoise_F32(mllib_f32 *gnoise, void *state,
 mllib_s32 n);
```

**Description** The `mllib_SignalGaussNoise_F32()` function generates one packet of Gaussian noise and updates the internal state.

**Parameters** The function takes the following arguments:

*gnoise*      Generated Gaussian noise array.

*state*        Internal state structure.

*n*            Length of the generated Gaussian wave array in number of samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGaussNoiseFree\_F32 – Gaussian noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalGaussNoiseFree_F32(void *state);
```

**Description** The `mllib_SignalGaussNoiseFree_F32()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGaussNoiseFree\_S16 – Gaussian noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalGaussNoiseFree_S16(void *state);
```

**Description** The `mllib_SignalGaussNoiseFree_S16()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGaussNoise\\_S16\(3MLIB\)](#), [mllib\\_SignalGaussNoiseInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalGaussNoiseInit\_F32 – Gaussian noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalGaussNoiseInit_F32(void **state, mllib_f32 mag,
 mllib_f32 mean, mllib_f32 stddev, mllib_f32 seed);
```

**Description** The `mllib_SignalGaussNoiseInit_F32()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

**Parameters** The function takes the following arguments:

- state* Internal state structure.
- mag* Magnitude of the Gaussian noise to be generated, in Q15 format.
- mean* Mean of the Gaussian noise.
- stddev* Standard deviation of the Gaussian noise.
- seed* Seed value for the pseudorandom number generator.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGaussNoiseInit\_S16 – Gaussian noise generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalGaussNoiseInit_S16(void **state, mllib_s16 mag,
 mllib_f32 mean, mllib_f32 stddev, mllib_s16 seed);
```

**Description** The `mllib_SignalGaussNoiseInit_S16()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

**Parameters** The function takes the following arguments:

*state* Internal state structure.

*mag* Magnitude of the Gaussian noise to be generated, in Q15 format.

*mean* Mean of the Gaussian noise.

*stddev* Standard deviation of the Gaussian noise.

*seed* Seed value for the pseudorandom number generator.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_SignalGaussNoise_S16(3MLIB)`, `mllib_SignalGaussNoiseFree_S16(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalGaussNoise\_S16 – Gaussian noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalGaussNoise_S16(mllib_s16 *gnoise, void *state,
mllib_s32 n);
```

**Description** The mllib\_SignalGaussNoise\_S16() function generates one packet of Gaussian noise and updates the internal state.

**Parameters** The function takes the following arguments:

*gnoise*      Generated Gaussian noise array.

*state*        Internal state structure.

*n*            Length of the generated Gaussian wave array in number of samples.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGaussNoiseFree\\_S16\(3MLIB\)](#), [mllib\\_SignalGaussNoiseInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalGenBartlett\_F32 – Bartlett window generation

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalGenBartlett_F32(mlib_f32 *window, mlib_s32 n);
```

**Description** The `mlib_SignalGenBartlett_F32()` function generates the normalized coefficients of the Bartlett window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array.

*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGenBartlett\_S16 – Bartlett window generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_SignalGenBartlett_S16(mllib_s16 *window, mllib_s32 n);`

**Description** The `mllib_SignalGenBartlett_S16()` function generates the normalized coefficients of the Bartlett window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.  
*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGenHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalGenHamming\\_S16\(3MLIB\)](#),  
[mllib\\_SignalGenBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalGenKaiser\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)



**Name** mllib\_SignalGenBlackman\_F32 – Blackman window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalGenBlackman_F32(mllib_f32 *window, mllib_f32 alpha,
 mllib_s32 n);
```

**Description** The `mllib_SignalGenBlackman_F32()` function generates the normalized coefficients of the Blackman window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array.

*alpha*        Blackman window parameter.  $-1 < \alpha < 0$ .

*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGenBlackman\_S16 – Blackman window generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalGenBlackman_S16(mllib_s16 *window, mllib_f32 alpha,
 mllib_s32 n);
```

**Description** The `mllib_SignalGenBlackman_S16()` function generates the normalized coefficients of the Blackman window.

**Parameters** The function takes the following arguments:

- window*      Generated window coefficient array. The window coefficients are in Q15 format.
- alpha*      Blackman window parameter.  $-1 < \alpha < 0$ .
- n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGenBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalGenHanning\\_S16\(3MLIB\)](#),  
[mllib\\_SignalGenHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalGenKaiser\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalGenHamming\_F32 – Hamming window generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_SignalGenHamming_F32(mllib_f32 *window, mllib_s32 n);`

**Description** The `mllib_SignalGenHamming_F32()` function generates the normalized coefficients of the Hamming window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array.

*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGenHamming\_S16 – Hamming window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalGenHamming_S16(mllib_s16 *window, mllib_s32 n);
```

**Description** The `mllib_SignalGenHamming_S16()` function generates the normalized coefficients of the Hamming window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.

*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGenBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalGenHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalGenBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalGenKaiser\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalGenHanning\_F32 – Hanning window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalGenHanning_F32(mllib_f32 *window, mllib_s32 n);
```

**Description** The `mllib_SignalGenHanning_F32()` function generates the normalized coefficients of the Hanning window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.  
*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGenHanning\_S16 – Hanning window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalGenHanning_S16(mllib_s16 *window, mllib_s32 n);
```

**Description** The `mllib_SignalGenHanning_S16()` function generates the normalized coefficients of the Hanning window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.  
*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGenBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalGenHamming\\_S16\(3MLIB\)](#),  
[mllib\\_SignalGenBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalGenKaiser\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalGenKaiser\_F32 – Kaiser window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalGenKaiser_F32(mllib_f32 *window, mllib_f32 beta,
mllib_s32 n);
```

**Description** The `mllib_SignalGenKaiser_F32()` function generates the normalized coefficients of the Kaiser window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.

*beta*         Kaiser window parameter.

*n*             Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalGenKaiser\_S16 – Kaiser window generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalGenKaiser_S16(mllib_s16 *window, mllib_f32 beta,
 mllib_s32 n);
```

**Description** The `mllib_SignalGenKaiser_S16()` function generates the normalized coefficients of the Kaiser window.

**Parameters** The function takes the following arguments:

*window*      Generated window coefficient array. The window coefficients are in Q15 format.

*beta*         Kaiser window parameter.

*n*            Length of window array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalGenBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalGenHanning\\_S16\(3MLIB\)](#),  
[mllib\\_SignalGenHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalGenBlackman\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)



**Name** mlib\_SignalIFFT\_1, mlib\_SignalIFFT\_1\_S16\_S16, mlib\_SignalIFFT\_1\_S16C\_S16C, mlib\_SignalIFFT\_1\_S16\_S16C, mlib\_SignalIFFT\_1\_S16, mlib\_SignalIFFT\_1\_S16C, mlib\_SignalIFFT\_1\_F32\_F32, mlib\_SignalIFFT\_1\_F32C\_F32C, mlib\_SignalIFFT\_1\_F32\_F32C, mlib\_SignalIFFT\_1\_F32, mlib\_SignalIFFT\_1\_F32C, mlib\_SignalIFFT\_1\_D64\_D64, mlib\_SignalIFFT\_1\_D64C\_D64C, mlib\_SignalIFFT\_1\_D64\_D64C, mlib\_SignalIFFT\_1\_D64, mlib\_SignalIFFT\_1\_D64C – signal Inverse Fast Fourier Transform (IFFT)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalIFFT_1_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti,
 const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_S16C_S16C(mlib_s16 *dstc,
 const mlib_s16 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_S16_S16C(mlib_s16 *dstr,
 const mlib_s16 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_S16(mlib_s16 *srcdstr,
 mlib_s16 *srcdsti,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_S16C(mlib_s16 *srcdstc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_F32_F32(mlib_f32 *dstr,
 mlib_f32 *dsti, const mlib_f32 *srcr,
 const mlib_f32 *srci, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_F32C_F32C(mlib_f32 *dstc,
 const mlib_f32 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_F32_F32C(mlib_f32 *dstr,
 const mlib_f32 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_F32(mlib_f32 *srcdstr,
 mlib_f32 *srcdsti,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_F32C(mlib_f32 *srcdstc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_1_D64_D64(mlib_d64 *dstr,
 mlib_d64 *dsti,
 const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);
```

```

mllib_status mllib_SignalIFFT_1_D64C_D64C(mlib_d64 *dstc,
 const mlib_d64 *srcc,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_1_D64_D64C(mlib_d64 *dstr,
 const mlib_d64 *srcc,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_1_D64(mlib_d64 *srcdstr,
 mlib_d64 *srcdsti,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_1_D64C(mlib_d64 *srcdstc,
 mlib_s32 order);

```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

$$dst[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{src[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$dst[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{src[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}order$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```

mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                                        |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                                   |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                             |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                                        |
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                  |
| <i>srcc</i>    | Complex source signal array. <code>srcc[2*i]</code> contains the real parts, and <code>srcc[2*i+1]</code> contains the imaginary parts.                       |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                             |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                        |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                              |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFT\\_2\(3MLIB\)](#), [mllib\\_SignalIFFT\\_3\(3MLIB\)](#), [mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIFFT\_2, mllib\_SignalIFFT\_2\_S16\_S16\_Mod,  
mllib\_SignalIFFT\_2\_S16C\_S16C\_Mod, mllib\_SignalIFFT\_2\_S16\_S16C\_Mod,  
mllib\_SignalIFFT\_2\_S16\_Mod, mllib\_SignalIFFT\_2\_S16C\_Mod,  
mllib\_SignalIFFT\_2\_F32\_F32, mllib\_SignalIFFT\_2\_F32C\_F32C,  
mllib\_SignalIFFT\_2\_F32\_F32C, mllib\_SignalIFFT\_2\_F32, mllib\_SignalIFFT\_2\_F32C,  
mllib\_SignalIFFT\_2\_D64\_D64, mllib\_SignalIFFT\_2\_D64C\_D64C,  
mllib\_SignalIFFT\_2\_D64\_D64C, mllib\_SignalIFFT\_2\_D64, mllib\_SignalIFFT\_2\_D64C –  
signal Inverse Fast Fourier Transform (IFFT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalIFFT_2_S16_S16_Mod(mllib_s16 *dstr,
 mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, mllib_s32 order);

mllib_status mllib_SignalIFFT_2_S16C_S16C_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_S16_S16C_Mod(mllib_s16 *dstr,
 const mllib_s16 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_S16_Mod(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_S16C_Mod(mllib_s16 *srcdstc, mllib_s32 order);

mllib_status mllib_SignalIFFT_2_F32_F32(mllib_f32 *dstr,
 mllib_f32 *dsti,
 const mllib_f32 *srcr, const mllib_f32 *srci, mllib_s32 order);

mllib_status mllib_SignalIFFT_2_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_F32_F32C(mllib_f32 *dstr,
 const mllib_f32 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_F32(mllib_f32 *srcdstr,
 mllib_f32 *srcdsti,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_F32C(mllib_f32 *srcdstc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_2_D64_D64(mllib_d64 *dstr,
 mllib_d64 *dsti,
 const mllib_d64 *srcr, const mllib_d64 *srci, mllib_s32 order);
```

```

mllib_status mllib_SignalIFFT_2_D64C_D64C(mlib_d64 *dstc,
 const mlib_d64 *srcc,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_2_D64_D64C(mlib_d64 *dstr,
 const mlib_d64 *srcc,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_2_D64(mlib_d64 *srcdstr,
 mlib_d64 *srcdsti,
 mlib_s32 order);

mllib_status mllib_SignalIFFT_2_D64C(mlib_d64 *srcdstc,
 mlib_s32 order);

```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

$$dst[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{src[n] * \exp(-j2*PI*n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$dst[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{src[k] * \exp(j2*PI*n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**}order$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```

mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- dstr* Destination signal array that contains the real parts.
- dsti* Destination signal array that contains the imaginary parts.
- srcr* Source signal array that contains the real parts.
- srci* Source signal array that contains the imaginary parts.
- dstc* Complex destination signal array. *dstc[2\*i]* contains the real parts, and *dstc[2\*i+1]* contains the imaginary parts.
- srcrcc* Complex source signal array. *srcrcc[2\*i]* contains the real parts, and *srcrcc[2\*i+1]* contains the imaginary parts.
- srcdstr* Source and destination signal array that contains the real parts.
- srcdsti* Source and destination signal array that contains the imaginary parts.
- srcdstc* Complex source and destination signal array. *srcdstc[2\*i]* contains the real parts, and *srcdstc[2\*i+1]* contains the imaginary parts.
- order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFT\\_1\(3MLIB\)](#), [mllib\\_SignalIFFT\\_3\(3MLIB\)](#), [mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalIFFT\_3, mlib\_SignalIFFT\_3\_S16\_S16\_Mod,  
 mlib\_SignalIFFT\_3\_S16C\_S16C\_Mod, mlib\_SignalIFFT\_3\_S16\_S16C\_Mod,  
 mlib\_SignalIFFT\_3\_S16\_Mod, mlib\_SignalIFFT\_3\_S16C\_Mod,  
 mlib\_SignalIFFT\_3\_F32\_F32, mlib\_SignalIFFT\_3\_F32C\_F32C,  
 mlib\_SignalIFFT\_3\_F32\_F32C, mlib\_SignalIFFT\_3\_F32, mlib\_SignalIFFT\_3\_F32C,  
 mlib\_SignalIFFT\_3\_D64\_D64, mlib\_SignalIFFT\_3\_D64C\_D64C,  
 mlib\_SignalIFFT\_3\_D64\_D64C, mlib\_SignalIFFT\_3\_D64, mlib\_SignalIFFT\_3\_D64C –  
 signal Inverse Fast Fourier Transform (IFFT)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalIFFT_3_S16_S16_Mod(mlib_s16 *dstr, mlib_s16 *dsti,
 const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_S16C_S16C_Mod(mlib_s16 *dstc,
 const mlib_s16 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_S16_S16C_Mod(mlib_s16 *dstr,
 const mlib_s16 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_S16_Mod(mlib_s16 *srcdstr,
 mlib_s16 *srcdsti,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_S16C_Mod(mlib_s16 *srcdstc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_F32_F32(mlib_f32 *dstr,
 mlib_f32 *dsti,
 const mlib_f32 *srcr, const mlib_f32 *srci,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_F32C_F32C(mlib_f32 *dstc,
 const mlib_f32 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_F32_F32C(mlib_f32 *dstr,
 const mlib_f32 *srcc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_F32(mlib_f32 *srcdstr,
 mlib_f32 *srcdsti,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_F32C(mlib_f32 *srcdstc,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFT_3_D64_D64(mlib_d64 *dstr,
 mlib_d64 *dsti,
 const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);
```

```

mllib_status mllib_SignalIFFT_3_D64C_D64C(mllib_d64 *dstc,
 const mllib_d64 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_3_D64_D64C(mllib_d64 *dstr,
 const mllib_d64 *srcc,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_3_D64(mllib_d64 *srcdstr,
 mllib_d64 *srcdsti,
 mllib_s32 order);

mllib_status mllib_SignalIFFT_3_D64C(mllib_d64 *srcdstc, mllib_s32 order);

```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

$$dst[k] = \frac{1}{C1} \sum_{n=0}^{N-1} \{src[n] * \exp(-j2\pi n*k/N)\}$$

and the following equation is used for inverse FFT (IFFT):

$$dst[n] = \frac{1}{C2} \sum_{k=0}^{N-1} \{src[k] * \exp(j2\pi n*k/N)\}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{**order}$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```

mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.



For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                                        |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                                   |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                             |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                                        |
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                  |
| <i>src</i>     | Complex source signal array. <code>src[2*i]</code> contains the real parts, and <code>src[2*i+1]</code> contains the imaginary parts.                         |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                             |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                        |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                              |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFT\\_1\(3MLIB\)](#), [mllib\\_SignalIFFT\\_2\(3MLIB\)](#), [mllib\\_SignalIFFT\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIFFT\_4, mllib\_SignalIFFT\_4\_S16\_S16, mllib\_SignalIFFT\_4\_S16C\_S16C, mllib\_SignalIFFT\_4\_S16\_S16C, mllib\_SignalIFFT\_4\_S16, mllib\_SignalIFFT\_4\_S16C – signal Inverse Fast Fourier Transform (IFFT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalIFFT_4_S16_S16(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci,
 mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFT_4_S16C_S16C(mllib_s16 *dstc,
 const mllib_s16 *srcr,
 mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFT_4_S16_S16C(mllib_s16 *dstr,
 const mllib_s16 *srcr,
 mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFT_4_S16(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti,
 mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFT_4_S16C(mllib_s16 *srcdstc,
 mllib_s32 order,
 mllib_s32 *scale);
```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] * \exp(-j2\pi n*k/N) \quad \text{C1}$$

and the following equation is used for inverse FFT (IFFT):

$$\text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] * \exp(j2\pi n*k/N) \quad \text{C2}$$

where

$k = 0, 1, \dots, (N - 1)$   
 $n = 0, 1, \dots, (N - 1)$   
 $N = 2^{*order}$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For `ScaleMode = 1`,  $C1 = 1$  and  $C2 = 2^{**order}$ .
- For `ScaleMode = 2`,  $C1 = 2^{**order}$  and  $C2 = 1$ .
- For `ScaleMode = 3`,  $C1 = C2 = 2^{**(\text{order}/2)}$  when `order` is even, or  $C1 = 2^{**((\text{order}+1)/2)}$  and  $C2 = 2^{**((\text{order}-1)/2)}$  when `order` is odd.
- For `ScaleMode = 4`,  $C1 = 2^{**P}$  and  $C2 = 2^{**Q}$ , where  $P$  and  $Q$  are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                                        |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                                   |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                             |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                                        |
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                  |
| <i>src</i>     | Complex source signal array. <code>src[2*i]</code> contains the real parts, and <code>src[2*i+1]</code> contains the imaginary parts.                         |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                             |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                        |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                              |
| <i>scale</i>   | Adaptive scaling factor.                                                                                                                                      |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_SignalFFT_1(3MLIB)`, `mllib_SignalFFT_2(3MLIB)`, `mllib_SignalFFT_3(3MLIB)`,  
`mllib_SignalFFT_4(3MLIB)`, `mllib_SignalIFFT_1(3MLIB)`, `mllib_SignalIFFT_2(3MLIB)`,  
`mllib_SignalIFFT_3(3MLIB)`, `attributes(5)`

**Name** mlib\_SignalIFFTW\_1, mlib\_SignalIFFTW\_1\_S16\_S16, mlib\_SignalIFFTW\_1\_S16C\_S16C, mlib\_SignalIFFTW\_1\_S16\_S16C, mlib\_SignalIFFTW\_1\_S16, mlib\_SignalIFFTW\_1\_S16C, mlib\_SignalIFFTW\_1\_F32\_F32, mlib\_SignalIFFTW\_1\_F32C\_F32C, mlib\_SignalIFFTW\_1\_F32\_F32C, mlib\_SignalIFFTW\_1\_F32, mlib\_SignalIFFTW\_1\_F32C – signal Inverse Fast Fourier Transform with windowing (IFFTW)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalIFFTW_1_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti,
 const mlib_s16 *srcr, const mlib_s16 *srci, const mlib_s16 *window,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_S16C_S16C(mlib_s16 *dstc,
 const mlib_s16 *srcr,
 const mlib_s16 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_S16_S16C(mlib_s16 *dstr,
 const mlib_s16 *srcr,
 const mlib_s16 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_S16(mlib_s16 *srcdstr, mlib_s16 *srcdsti,
 const mlib_s16 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_S16C(mlib_s16 *srcdstc,
 const mlib_s16 *window,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_F32_F32(mlib_f32 *dstr,
 mlib_f32 *dsti,
 const mlib_f32 *srcr, const mlib_f32 *srci, const mlib_f32 *window,
 mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_F32C_F32C(mlib_f32 *dstc,
 const mlib_f32 *srcr,
 const mlib_f32 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_F32_F32C(mlib_f32 *dstr,
 const mlib_f32 *srcr,
 const mlib_f32 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_F32(mlib_f32 *srcdstr, mlib_f32 *srcdsti,
 const mlib_f32 *window, mlib_s32 order);
```

```
mlib_status mlib_SignalIFFTW_1_F32C(mlib_f32 *srcdstc,
 const mlib_f32 *window,
 mlib_s32 order);
```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \text{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)\}$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \text{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$$\begin{aligned} k &= 0, 1, \dots, (N - 1) \\ n &= 0, 1, \dots, (N - 1) \\ N &= 2^{**}\text{order} \end{aligned}$$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i> | Destination signal array that contains the real parts.                                                                             |
| <i>dsti</i> | Destination signal array that contains the imaginary parts.                                                                        |
| <i>srcr</i> | Source signal array that contains the real parts.                                                                                  |
| <i>srci</i> | Source signal array that contains the imaginary parts.                                                                             |
| <i>dstc</i> | Complex destination signal array. <i>dstc</i> [2*i] contains the real parts, and <i>dstc</i> [2*i+1] contains the imaginary parts. |
| <i>srcc</i> | Complex source signal array. <i>srcc</i> [2*i] contains the real parts, and <i>srcc</i> [2*i+1] contains the imaginary parts.      |

- srcdstr* Source and destination signal array that contains the real parts.
- srcdsti* Source and destination signal array that contains the imaginary parts.
- srcdstc* Complex source and destination signal array. *srcdstc*[2\*i] contains the real parts, and *srcdstc*[2\*i+1] contains the imaginary parts.
- window* Window coefficient array with 2\*\*order real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.
- order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)





## REFERENCE

### Multimedia Library Functions - Part 5

**Name** mllib\_SignalIFFTW\_2, mllib\_SignalIFFTW\_2\_S16\_S16\_Mod,  
mllib\_SignalIFFTW\_2\_S16C\_S16C\_Mod, mllib\_SignalIFFTW\_2\_S16\_S16C\_Mod,  
mllib\_SignalIFFTW\_2\_S16\_Mod, mllib\_SignalIFFTW\_2\_S16C\_Mod,  
mllib\_SignalIFFTW\_2\_F32\_F32, mllib\_SignalIFFTW\_2\_F32C\_F32C,  
mllib\_SignalIFFTW\_2\_F32\_F32C, mllib\_SignalIFFTW\_2\_F32, mllib\_SignalIFFTW\_2\_F32C –  
signal Inverse Fast Fourier Transform with windowing (IFFTW)

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalIFFTW_2_S16_S16_Mod(mllib_s16 *dstr,
 mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_S16C_S16C_Mod(mllib_s16 *dstc,
 const mllib_s16 *srcc,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_S16_S16C_Mod(mllib_s16 *dstr,
 const mllib_s16 *srcc,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_S16_Mod(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti,
 const mllib_s16 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_S16C_Mod(mllib_s16 *srcdstc,
 const mllib_s16 *window,
 mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_F32_F32(mllib_f32 *dstr,
 mllib_f32 *dsti,
 const mllib_f32 *srcr, const mllib_f32 *srci, const mllib_f32 *window,
 mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_F32C_F32C(mllib_f32 *dstc,
 const mllib_f32 *srcc,
 const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_F32_F32C(mllib_f32 *dstr,
 const mllib_f32 *srcc,
 const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_F32(mllib_f32 *srcdstr,
 mllib_f32 *srcdsti,
 const mllib_f32 *window, mllib_s32 order);

mllib_status mllib_SignalIFFTW_2_F32C(mllib_f32 *srcdstc,
 const mllib_f32 *window,
 mllib_s32 order);
```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \text{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)\}$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \text{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$$\begin{aligned} k &= 0, 1, \dots, (N - 1) \\ n &= 0, 1, \dots, (N - 1) \\ N &= 2^{**}\text{order} \end{aligned}$$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

*dstr* Destination signal array that contains the real parts.  
*dsti* Destination signal array that contains the imaginary parts.  
*srcr* Source signal array that contains the real parts.  
*srci* Source signal array that contains the imaginary parts.

- dstc* Complex destination signal array. *dstc*[2\*i] contains the real parts, and *dstc*[2\*i+1] contains the imaginary parts.
- srcc* Complex source signal array. *srcc*[2\*i] contains the real parts, and *srcc*[2\*i+1] contains the imaginary parts.
- srcdstr* Source and destination signal array that contains the real parts.
- srcdsti* Source and destination signal array that contains the imaginary parts.
- srcdstc* Complex source and destination signal array. *srcdstc*[2\*i] contains the real parts, and *srcdstc*[2\*i+1] contains the imaginary parts.
- window* Window coefficient array with 2\*\*order real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.
- order* Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalFFTW\\_4\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIFFTW\_3, mllib\_SignalIFFTW\_3\_S16\_S16\_Mod, mllib\_SignalIFFTW\_3\_S16C\_S16C\_Mod, mllib\_SignalIFFTW\_3\_S16\_S16C\_Mod, mllib\_SignalIFFTW\_3\_S16\_Mod, mllib\_SignalIFFTW\_3\_S16C\_Mod, mllib\_SignalIFFTW\_3\_F32\_F32, mllib\_SignalIFFTW\_3\_F32C\_F32C, mllib\_SignalIFFTW\_3\_F32\_F32C, mllib\_SignalIFFTW\_3\_F32, mllib\_SignalIFFTW\_3\_F32C – signal Inverse Fast Fourier Transform with windowing (IFFTW)

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalIFFTW_3_S16_S16_Mod(mllib_s16 *dst, mllib_s16 *dsti,
 const mllib_s16 *src, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_S16C_S16C_Mod(mllib_s16 *dst,
 const mllib_s16 *src,
 const mllib_s16 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_S16_S16C_Mod(mllib_s16 *dst,
 const mllib_s16 *src,
 const mllib_s16 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_S16_Mod(mllib_s16 *srcdst,
 mllib_s16 *srcdsti,
 const mllib_s16 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_S16C_Mod(mllib_s16 *srcdst,
 const mllib_s16 *window,
 mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_F32_F32(mllib_f32 *dst,
 mllib_f32 *dsti,
 const mllib_f32 *src, const mllib_f32 *srci, const mllib_f32 *window,
 mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_F32C_F32C(mllib_f32 *dst,
 const mllib_f32 *src,
 const mllib_f32 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_F32_F32C(mllib_f32 *dst,
 const mllib_f32 *src,
 const mllib_f32 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_F32(mllib_f32 *srcdst,
 mllib_f32 *srcdsti,
 const mllib_f32 *window, mllib_s32 order);
```

```
mllib_status mllib_SignalIFFTW_3_F32C(mllib_f32 *srcdst,
 const mllib_f32 *window,
 mllib_s32 order);
```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \text{C1} \sum_{n=0}^{N-1} \{\text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)\}$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \text{C2} \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$$\begin{aligned} k &= 0, 1, \dots, (N - 1) \\ n &= 0, 1, \dots, (N - 1) \\ N &= 2^{**}\text{order} \end{aligned}$$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2\*\*order.
- For ScaleMode = 2, C1 = 2\*\*order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2\*\*((order/2) when order is even, or C1 = 2\*\*((order+1)/2) and C2 = 2\*\*((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2\*\*P and C2 = 2\*\*Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

*dstr* Destination signal array that contains the real parts.

*dsti* Destination signal array that contains the imaginary parts.

*srrc* Source signal array that contains the real parts.

*srcl* Source signal array that contains the imaginary parts.

|                |                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                          |
| <i>srcc</i>    | Complex source signal array. <code>srcc[2*i]</code> contains the real parts, and <code>srcc[2*i+1]</code> contains the imaginary parts.                               |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                                     |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                                |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts.         |
| <i>window</i>  | Window coefficient array with $2^{**}order$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                                      |

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFT\\_1\(3MLIB\)](#), [mllib\\_SignalFFT\\_2\(3MLIB\)](#), [mllib\\_SignalFFT\\_3\(3MLIB\)](#), [mllib\\_SignalFFT\\_4\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIFFTW\_4, mllib\_SignalIFFTW\_4\_S16\_S16, mllib\_SignalIFFTW\_4\_S16C\_S16C, mllib\_SignalIFFTW\_4\_S16\_S16C, mllib\_SignalIFFTW\_4\_S16, mllib\_SignalIFFTW\_4\_S16C – signal Inverse Fast Fourier Transform with windowing (IFFTW)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalIFFTW_4_S16_S16(mllib_s16 *dstr, mllib_s16 *dsti,
 const mllib_s16 *srcr, const mllib_s16 *srci, const mllib_s16 *window,
 mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFTW_4_S16C_S16C(mllib_s16 *dstc,
 const mllib_s16 *srcr,
 const mllib_s16 *window, mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFTW_4_S16_S16C(mllib_s16 *dstr,
 const mllib_s16 *srcr,
 const mllib_s16 *window, mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFTW_4_S16(mllib_s16 *srcdstr,
 mllib_s16 *srcdsti,
 const mllib_s16 *window, mllib_s32 order, mllib_s32 *scale);
```

```
mllib_status mllib_SignalIFFTW_4_S16C(mllib_s16 *srcdstc,
 const mllib_s16 *window,
 mllib_s32 order, mllib_s32 *scale);
```

**Description** Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

$$\text{dst}[k] = \sum_{n=0}^{N-1} \{\text{src}[n] * \text{window}[n] * \exp(-j2\pi n*k/N)\}$$

and the IFFTW functions use the following equation:

$$\text{dst}[n] = \sum_{k=0}^{N-1} \{\text{src}[k] * \text{window}[k] * \exp(j2\pi n*k/N)\}$$

where

$$\begin{aligned} k &= 0, 1, \dots, (N - 1) \\ n &= 0, 1, \dots, (N - 1) \\ N &= 2^{**order} \end{aligned}$$

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mllib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mllib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```



The scaling factors  $C1$  and  $C2$  used in the equations are defined as follows:

- For `ScaleMode = 1`,  $C1 = 1$  and  $C2 = 2^{**}order$ .
- For `ScaleMode = 2`,  $C1 = 2^{**}order$  and  $C2 = 1$ .
- For `ScaleMode = 3`,  $C1 = C2 = 2^{**}(order/2)$  when `order` is even, or  $C1 = 2^{**}((order+1)/2)$  and  $C2 = 2^{**}((order-1)/2)$  when `order` is odd.
- For `ScaleMode = 4`,  $C1 = 2^{**}P$  and  $C2 = 2^{**}Q$ , where  $P$  and  $Q$  are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

|                |                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstr</i>    | Destination signal array that contains the real parts.                                                                                                                |
| <i>dsti</i>    | Destination signal array that contains the imaginary parts.                                                                                                           |
| <i>srcr</i>    | Source signal array that contains the real parts.                                                                                                                     |
| <i>srci</i>    | Source signal array that contains the imaginary parts.                                                                                                                |
| <i>dstc</i>    | Complex destination signal array. <code>dstc[2*i]</code> contains the real parts, and <code>dstc[2*i+1]</code> contains the imaginary parts.                          |
| <i>src</i>     | Complex source signal array. <code>src[2*i]</code> contains the real parts, and <code>src[2*i+1]</code> contains the imaginary parts.                                 |
| <i>srcdstr</i> | Source and destination signal array that contains the real parts.                                                                                                     |
| <i>srcdsti</i> | Source and destination signal array that contains the imaginary parts.                                                                                                |
| <i>srcdstc</i> | Complex source and destination signal array. <code>srcdstc[2*i]</code> contains the real parts, and <code>srcdstc[2*i+1]</code> contains the imaginary parts.         |
| <i>window</i>  | Window coefficient array with $2^{**}order$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type. |
| <i>order</i>   | Order of the transformation. The base-2 logarithm of the number of data samples.                                                                                      |
| <i>scale</i>   | Adaptive scaling factor.                                                                                                                                              |

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalFFTW\\_3\(3MLIB\)](#), [mllib\\_SignalFFTW\\_4\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_1\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_2\(3MLIB\)](#), [mllib\\_SignalIFFTW\\_3\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIR\_Biquad\_S16\_S16\_Sat, mllib\_SignalIIR\_Biquad\_S16S\_S16S\_Sat, mllib\_SignalIIR\_Biquad\_F32\_F32, mllib\_SignalIIR\_Biquad\_F32S\_F32S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalIIR_Biquad_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalIIR_Biquad_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

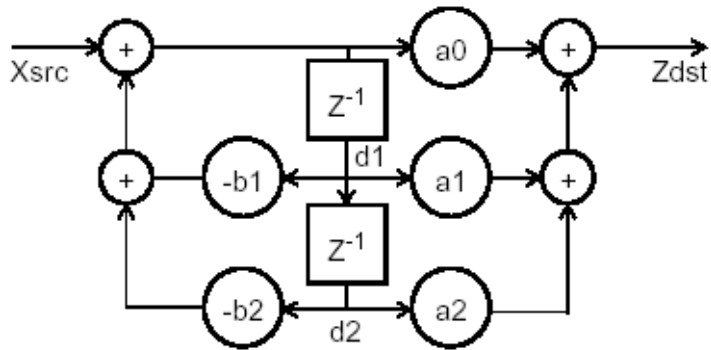
mllib_status mllib_SignalIIR_Biquad_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalIIR_Biquad_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);
```

**Description** Each of these functions applies a biquad IIR filter to a signal array.

$$\begin{aligned}
 X &= x(n) \quad n = 0, 1, \dots \\
 Z &= z(n) \quad n = 0, 1, \dots \\
 &= \sum_{k=0}^N a_k x(n-k) + \sum_{k=1}^M b_k z(n-k) \quad n = 0, 1, \dots \\
 H(z) &= \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}
 \end{aligned}$$

The biquad IIR filter is represented by the following figure:



**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src* Source signal array.
- filter* Internal filter structure.
- n* Number of samples in the source signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalIIRFree\_Biquad\_F32\_F32, mlib\_SignalIIRFree\_Biquad\_F32S\_F32S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
void mlib_SignalIIRFree_Biquad_F32_F32(void *filter);
void mlib_SignalIIRFree_Biquad_F32S_F32S(void *filter);
```

**Description** Each of these functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes the following arguments:

*filter*     Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mlib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#), [mlib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIRFree\_Biquad\_S16\_S16, mllib\_SignalIIRFree\_Biquad\_S16S\_S16S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalIIRFree_Biquad_S16_S16(void *filter);
void mllib_SignalIIRFree_Biquad_S16S_S16S(void *filter);
```

**Description** Each of these functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#), [mllib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIRFree\_P4\_F32\_F32, mllib\_SignalIIRFree\_P4\_F32S\_F32S – parallel Infinite Impulse Response (IIR) filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`  
  
`void mllib_SignalIIRFree_P4_F32_F32(void *filter);`  
`void mllib_SignalIIRFree_P4_F32S_F32S(void *filter);`

**Description** Each of these functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes the following arguments:  
  
*filter*      Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIRFree\_P4\_S16\_S16, mllib\_SignalIIRFree\_P4\_S16S\_S16S – parallel Infinite Impulse Response (IIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalIIRFree_P4_S16_S16(void *filter);
void mllib_SignalIIRFree_P4_S16S_S16S(void *filter);
```

**Description** Each of these functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalIIRInit\_Biquad\_F32\_F32, mllib\_SignalIIRInit\_Biquad\_F32S\_F32S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalIIRInit_Biquad_F32_F32(void **filter,
const mlib_f32 *flt);

mllib_status mllib_SignalIIRInit_Biquad_F32S_F32S(void **filter,
const mlib_f32 *flt);
```

**Description** Each of these functions allocates memory for the internal file structure and converts the filter coefficients into an internal representation.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

*flt* Array of five filter coefficients: a0, a1, a2, b1, and b2.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalIIRInit\_Biquad\_S16\_S16, mllib\_SignalIIRInit\_Biquad\_S16S\_S16S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mllib.h>

mllib_status mllib_SignalIIRInit_Biquad_S16_S16(void **filter,
const mlib_f32 *flt);

mllib_status mllib_SignalIIRInit_Biquad_S16S_S16S(void **filter,
const mlib_f32 *flt);
```

**Description** Each of these functions allocates memory for the internal file structure and converts the filter coefficients into an internal representation.

**Parameters** Each of the functions takes the following arguments:

- filter* Internal filter structure.
- flt* Array of five filter coefficients: a0, a1, a2, b1, and b2.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalIIRInit\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mlib\_SignalIIRInit\_P4\_F32\_F32, mlib\_SignalIIRInit\_P4\_F32S\_F32S – parallel Infinite Impulse Response (IIR) filtering

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalIIRInit_P4_F32_F32(void **filter,
 const mlib_f32 flt);
```

```
mlib_status mlib_SignalIIRInit_P4_F32S_F32S(void **filter,
 const mlib_f32 flt);
```

**Description** Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

**Parameters** Each of the functions takes the following arguments:

*filter* Internal filter structure.

*flt* Array of nine filter coefficients: c, a00, a10, b10, b20, a01, a11, b11, and b21.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mlib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mlib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIRInit\_P4\_S16\_S16, mllib\_SignalIIRInit\_P4\_S16S\_S16S – parallel Infinite Impulse Response (IIR) filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalIIRInit_P4_S16_S16(void **filter,
 const mllib_f32 flt);

mllib_status mllib_SignalIIRInit_P4_S16S_S16S(void **filter,
 const mllib_f32 flt);
```

**Description** Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

**Parameters** Each of the functions takes the following arguments:

*filter*     Internal filter structure.

*flt*        Array of nine filter coefficients: c, a00, a10, b10, b20, a01, a11, b11, and b21.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIIR\\_Biquad\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIR\\_P4\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_Biquad\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRFree\\_P4\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalIIRInit\\_Biquad\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIIR\_P4\_S16\_S16\_Sat, mllib\_SignalIIR\_P4\_S16S\_S16S\_Sat, mllib\_SignalIIR\_P4\_F32\_F32, mllib\_SignalIIR\_P4\_F32S\_F32S – parallel Infinite Impulse Response (IIR) filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalIIR_P4_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalIIR_P4_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *filter, mllib_s32 n);

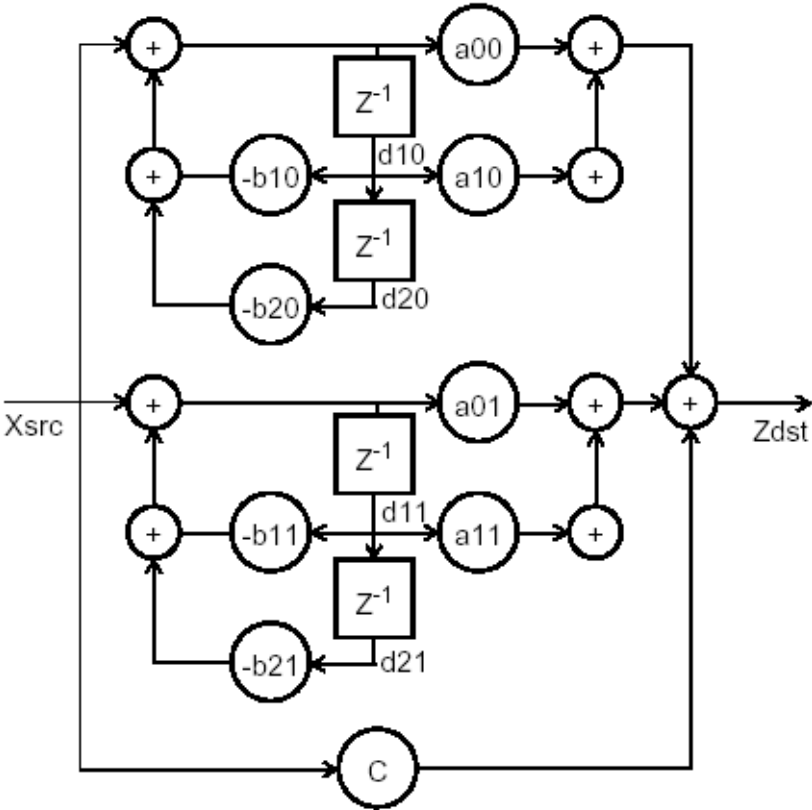
mllib_status mllib_SignalIIR_P4_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);

mllib_status mllib_SignalIIR_P4_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, void *filter, mllib_s32 n);
```

**Description** Each of these functions applies a fourth order parallel IIR filter to one signal packet and updates the filter state.

$$\begin{aligned}
 X &= x(n) \quad n = 0, 1, \dots \\
 Z &= z(n) \quad n = 0, 1, \dots \\
 &= \sum_{k=0}^N a_k x(n-k) + \sum_{k=1}^M b_k z(n-k) \quad n = 0, 1, \dots \\
 H(z) &= C + \sum_{k=0}^1 \frac{(a_{0k} + a_{1k}z^{-1})}{(1 + b_{1k}z^{-1} + b_{2k}z^{-2})}
 \end{aligned}$$

The fourth order parallel IIR filter is represented by the following figure:



**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src* Source signal array.
- filter* Internal filter structure.
- n* Number of signal samples.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** `mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),`  
`mlib_SignalIIRFree_Biquad_S16_S16(3MLIB),`  
`mlib_SignalIIRFree_P4_S16_S16(3MLIB),`  
`mlib_SignalIIRInit_Biquad_S16_S16(3MLIB),`  
`mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)`

**Name** mllib\_SignalIMDCT\_D64 – Dolby AC-3 digital audio standard transformation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalIMDCT_D64(mllib_d64 *data);
```

**Description** The `mllib_SignalIMDCT_D64()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

**Parameters** The function takes the following arguments:

*data*      Pointer to the data array. `data[2*i]` contains the real parts, and `data[2*i+1]` contains the imaginary parts.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIMDCT\\_F32\(3MLIB\)](#), [mllib\\_SignalIMDCTSplit\\_D64\(3MLIB\)](#), [mllib\\_SignalIMDCTSplit\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalIMDCT\_F32 – Dolby AC-3 digital audio standard transformation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalIMDCT_F32(mllib_f32 *data);
```

**Description** The `mllib_SignalIMDCT_F32()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

**Parameters** The function takes the following arguments:

*data*      Pointer to the data array. `data[2*i]` contains the real parts, and `data[2*i+1]` contains the imaginary parts.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIMDCT\\_D64\(3MLIB\)](#), [mllib\\_SignalIMDCTSplit\\_D64\(3MLIB\)](#), [mllib\\_SignalIMDCTSplit\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalIMDCTSplit\_D64 – Dolby AC-3 digital audio standard transformation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalIMDCTSplit_D64(mllib_d64 *data);
```

**Description** The `mllib_SignalIMDCTSplit_D64()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

**Parameters** The function takes the following arguments:

*data*      Pointer to the data array. `data[4*i]` contains the real parts of the first array, `data[4*i+1]` contains the real parts of the second array, `data[4*i+2]` contains the imaginary parts of the first array, and `data[4*i+3]` contains the imaginary parts of the second array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalIMDCT\\_D64\(3MLIB\)](#), [mllib\\_SignalIMDCT\\_F32\(3MLIB\)](#), [mllib\\_SignalIMDCTSplit\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalIMDCTSplit\_F32 – Dolby AC-3 digital audio standard transformation

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalIMDCTSplit_F32(mlib_f32 *data);
```

**Description** The `mlib_SignalIMDCTSplit_F32()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

**Parameters** The function takes the following arguments:

*data* Pointer to the data array. `data[4*i]` contains the real parts of the first array, `data[4*i+1]` contains the real parts of the second array, `data[4*i+2]` contains the imaginary parts of the first array, and `data[4*i+3]` contains the imaginary parts of the second array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalIMDCT\\_D64\(3MLIB\)](#), [mlib\\_SignalIMDCT\\_F32\(3MLIB\)](#), [mlib\\_SignalIMDCTSplit\\_D64\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLimit, mlib\_SignalLimit\_S16\_S16, mlib\_SignalLimit\_S16S\_S16S, mlib\_SignalLimit\_S16, mlib\_SignalLimit\_S16S, mlib\_SignalLimit\_F32\_F32, mlib\_SignalLimit\_F32S\_F32S, mlib\_SignalLimit\_F32, mlib\_SignalLimit\_F32S – signal hard limiting

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalLimit_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *low,
 const mlib_s16 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *low,
 const mlib_s16 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_S16(mlib_s16 *srcdst,
 const mlib_s16 *low, const mlib_s16 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_S16S(mlib_s16 *srcdst,
 const mlib_s16 *low, const mlib_s16 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *low,
 const mlib_f32 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *low,
 const mlib_f32 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_F32(mlib_f32 *srcdst,
 const mlib_f32 *low, const mlib_f32 *high, mlib_s32 n);

mlib_status mlib_SignalLimit_F32S(mlib_f32 *srcdst,
 const mlib_f32 *low, const mlib_f32 *high, mlib_s32 n);
```

**Description** Each of these functions performs hard limiting.

For monaural signals, the following equation is used:

$$\begin{array}{ll} \text{dst}[i] = \text{low}[0] & \text{if } \text{src}[i] < \text{low}[0] \\ \text{dst}[i] = \text{src}[i] & \text{if } \text{low}[0] \leq \text{src}[i] < \text{high}[0] \\ \text{dst}[i] = \text{high}[0] & \text{if } \text{src}[i] \geq \text{high}[0] \end{array}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For stereo signals, the following equation is used:

$$\begin{array}{ll} \text{dst}[2*i] = \text{low}[0] & \text{if } \text{src}[2*i] < \text{low}[0] \\ \text{dst}[2*i] = \text{src}[2*i] & \text{if } \text{low}[0] \leq \text{src}[2*i] < \text{high}[0] \\ \text{dst}[2*i] = \text{high}[0] & \text{if } \text{src}[2*i] \geq \text{high}[0] \\ \\ \text{dst}[2*i+1] = \text{low}[1] & \text{if } \text{src}[2*i+1] < \text{low}[1] \end{array}$$

```
dst[2*i+1] = src[2*i+1] if low[1] ≤ src[2*i+1] < high[1]
dst[2*i+1] = high[1] if src[2*i+1] ≥ high[1]
```

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes some of the following arguments:

*dst* Destination signal array.

*src* Source signal array.

*srcdst* Source and destination signal array.

*low* Lower input limit. In the stereo version, *low*[0] contains the lower limit for channel 0, and *low*[1] contains the lower limit for channel 1.

*high* Upper input limit. In the stereo version. *high*[0] contains the upper limit for channel 0, and *high*[1] contains the upper limit for channel 1.

*n* Number of samples in the source signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalLinear2ADPCM2Bits – adaptive differential pulse code modulation (ADPCM)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalLinear2ADPCM2Bits(mllib_u8 *adpcm,
 const mllib_s16 *pcm,void *state, mllib_s32 n);
```

**Description** The `mllib_SignalLinear2ADPCM2Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 2-bit ADPCM to 16-bit linear PCM to G.723 or G.726 16kbps format.

**Parameters** The function takes the following arguments:

- adpcm* ADPCM code array.
- pcm* Linear PCM sample array.
- state* Internal structure of the codec.
- n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalADPCM2Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM3Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM4Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM5Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCMFree\(3MLIB\)](#), [mllib\\_SignalADPCMInit\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM3Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM4Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM5Bits\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLinear2ADPCM3Bits – adaptive differential pulse code modulation (ADPCM)

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalLinear2ADPCM3Bits(mlib_u8 *adpcm,
 const mlib_s16 *pcm, void *state, mlib_s32 n);
```

**Description** The `mlib_SignalLinear2ADPCM3Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.723 or G.726 24kbps 3-bit ADPCM format.

**Parameters** The function takes the following arguments:

*adpcm* ADPCM code array.

*pcm* Linear PCM sample array.

*state* Internal structure of the codec.

*n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalLinear2ADPCM4Bits – adaptive differential pulse code modulation (ADPCM)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalLinear2ADPCM4Bits(mllib_u8 *adpcm,
 const mllib_s16 *pcm, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalLinear2ADPCM4Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.721 or G.726 32kbps 4-bit ADPCM format.

**Parameters** The function takes the following arguments:

- adpcm* ADPCM code array.
- pcm* Linear PCM sample array.
- state* Internal structure of the codec.
- n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalADPCM2Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM3Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM4Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCM5Bits2Linear\(3MLIB\)](#), [mllib\\_SignalADPCMFree\(3MLIB\)](#), [mllib\\_SignalADPCMInit\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM2Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM3Bits\(3MLIB\)](#), [mllib\\_SignalLinear2ADPCM5Bits\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalLinear2ADPCM5Bits – adaptive differential pulse code modulation (ADPCM)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalLinear2ADPCM5Bits(mlib_u8 *adpcm,
 const mlib_s16 *pcm, void *state, mlib_s32 n);
```

**Description** The `mlib_SignalLinear2ADPCM5Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.723 or G.726 40kbps 5-bit ADPCM format.

**Parameters** The function takes the following arguments:

*adpcm* ADPCM code array.  
*pcm* Linear PCM sample array.  
*state* Internal structure of the codec.  
*n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM3Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_SignalLinear2ALaw – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalLinear2ALaw(mllib_u8 *acode,
 const mllib_s16 *pcm, mllib_s32 n);
```

**Description** The `mllib_SignalLinear2ALaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

- acode*     A-law code array.
- pcm*       Linear PCM sample array.
- n*          Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalALaw2Linear\(3MLIB\)](#), [mllib\\_SignalALaw2uLaw\(3MLIB\)](#),  
[mllib\\_SignalLinear2uLaw\(3MLIB\)](#), [mllib\\_SignaluLaw2ALaw\(3MLIB\)](#),  
[mllib\\_SignaluLaw2Linear\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLinear2uLaw – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalLinear2uLaw(mlib_u8 *ucode,
 const mlib_s16 *pcm, mlib_s32 n);
```

**Description** The `mlib_SignalLinear2uLaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

*ucode* m-law code array.

*pcm* Linear PCM sample array.

*n* Number of samples in the source array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalALaw2Linear\(3MLIB\)](#), [mlib\\_SignalALaw2uLaw\(3MLIB\)](#), [mlib\\_SignalLinear2ALaw\(3MLIB\)](#), [mlib\\_SignaluLaw2ALaw\(3MLIB\)](#), [mlib\\_SignaluLaw2Linear\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLMSFilter, mllib\_SignalLMSFilterInit\_S16\_S16, mllib\_SignalLMSFilterInit\_S16S\_S16S, mllib\_SignalLMSFilterInit\_F32\_F32, mllib\_SignalLMSFilterInit\_F32S\_F32S, mllib\_SignalLMSFilter\_S16\_S16\_Sat, mllib\_SignalLMSFilter\_S16S\_S16S\_Sat, mllib\_SignalLMSFilter\_F32\_F32, mllib\_SignalLMSFilter\_F32S\_F32S, mllib\_SignalLMSFilterNonAdapt\_S16\_S16\_Sat, mllib\_SignalLMSFilterNonAdapt\_S16S\_S16S\_Sat, mllib\_SignalLMSFilterNonAdapt\_F32\_F32, mllib\_SignalLMSFilterNonAdapt\_F32S\_F32S, mllib\_SignalLMSFilterFree\_S16\_S16, mllib\_SignalLMSFilterFree\_S16S\_S16S, mllib\_SignalLMSFilterFree\_F32\_F32, mllib\_SignalLMSFilterFree\_F32S\_F32S – least mean square (LMS) adaptive filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalLMSFilterInit_S16_S16(void **filter,
 const mlib_f32 *flt,
 mlib_s32 tap, mlib_f32 beta);

mllib_status mllib_SignalLMSFilterInit_S16S_S16S(void **filter,
 const mlib_f32 *flt,
 mlib_s32 tap, mlib_f32 beta);

mllib_status mllib_SignalLMSFilterInit_F32_F32(void **filter,
 const mlib_f32 *flt,
 mlib_s32 tap, mlib_f32 beta);

mllib_status mllib_SignalLMSFilterInit_F32S_F32S(void **filter,
 const mlib_f32 *flt,
 mlib_s32 tap, mlib_f32 beta);

mllib_status mllib_SignalLMSFilter_S16_S16_Sat(mlib_s16 *dst,
 const mlib_s16 *src,
 const mlib_s16 *ref, void *filter, mlib_s32 n);

mllib_status mllib_SignalLMSFilter_S16S_S16S_Sat(mlib_s16 *dst,
 const mlib_s16 *src,
 const mlib_s16 *ref, void *filter, mlib_s32 n);

mllib_status mllib_SignalLMSFilter_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src,
 const mlib_f32 *ref, void *filter, mlib_s32 n);

mllib_status mllib_SignalLMSFilter_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src,
 const mlib_f32 *ref, void *filter, mlib_s32 n);

mllib_status mllib_SignalLMSFilterNonAdapt_S16_S16_Sat(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref,
 void *filter, mlib_s32 n);
```

```

mlib_status mlib_SignalLMSFilterNonAdapt_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref,
 void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilterNonAdapt_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref,
 void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilterNonAdapt_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref,
 void *filter, mlib_s32 n);

void mlib_SignalLMSFilterFree_S16_S16(void *filter);

void mlib_SignalLMSFilterFree_S16S_S16S(void *filter);

void mlib_SignalLMSFilterFree_F32_F32(void *filter);

void mlib_SignalLMSFilterFree_F32S_F32S(void *filter);

```

**Description** The basic LMS adaptive algorithm is summarized as follows:

1. Initialize the weights  $W_k(i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ .
2. Initialize previous source elements  $X_o(i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ .
3. Read  $X_k(t)$  from *src* and  $Y_k(t)$  from *ref*,  $t = 0, 1, \dots, n - 1$ .
4. Compute filter output:  $n_k = \sum(W_k(i) * X_k(t - i))$ ,  $i = 0, 1, \dots, \text{tap} - 1$ . If  $i > t$ , use previous source elements stored in the  $X_o$  vector.
5. Store filter output:  $\text{dst}[t] = n_k$ .
6. Compute the error estimate:  $E_k = Y_k - n_k$ .
7. Compute factor  $BE_0 = 2 * \text{beta} * E_k$ .
8. Update filter weights:  $W_k(i) += BE_0 * X_k(t - i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ . If  $i > t$ , use previous source elements stored in  $X_o$  vector.
9. Next  $t$ , go to step 3.
10. Store  $N$  ending source elements in previous source elements vector  $X_o$ : if  $N > n$ ,  $N = n$ ; else  $N = \text{tap}$ .

The functions assume that the input signal has a power maximum equal to 1. If it is not, *beta* should be divided by power maximum. Power maximum is calculated according to the following formula:

$$\text{Power\_max} = \text{MAX}_n \left\{ \sum_{k=0}^{\text{flt\_len}} \text{signal}(n + k)^2 \right\}$$

It is necessary to consider the maximum of power maxima of both components as the stereo signal's power maximum.

Each of the `FilterInit` functions allocates memory for the internal filter structure and converts the parameters into the internal representation.

Each of the `Filter` functions applies the LMS adaptive filter on one signal packet and updates the filter states.

Each of the `FilterNoAdapt` functions applies the LMS filter on one signal packet and updates the filter states but without changing the filter weights.

Each of the `FilterFree` functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes some of the following arguments:

- filter* Internal filter structure.
- flt* Filter coefficient array.
- tap* Taps of the filter.
- beta* Error weighting factor.  $0 < \text{beta} < 1$ .
- dst* Destination signal array.
- src* Source signal array.
- ref* Reference or “desired” signal array.
- n* Number of samples in the source signal array.

**Return Values** Each of the `FilterInit`, `Filter` and `FilterNonAdapt` functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. The `FilterFree` functions don't return anything.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalNLMSFilter\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPC2Cepstral\_F32 – convert linear prediction coefficients to cepstral coefficients

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalLPC2Cepstral_F32(mllib_f32 *cepst,
 const mllib_f32 *lpc, mllib_f32 gain, mllib_s32 length,
 mllib_s32 order);
```

**Description** The `mllib_SignalLPC2Cepstral_F32()` function converts linear prediction coefficients to cepstral coefficients.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

$$c(0) = \log(G)$$
$$c(m) = a(m) + \sum_{k=1}^{m-1} \frac{k}{m} * c(k) * a(m-k), \quad 1 \leq m \leq M$$
$$c(m) = \sum_{k=1}^{m-1} \frac{k}{m} * c(k) * a(m-k), \quad m > M$$

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Bing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*cepst*        The cepstral coefficients.

*lpc*         The linear prediction coefficients.

*gain*        The gain of the LPC model.

*length*      The length of the cepstral coefficients.

*order*       The order of the linear prediction filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mllib\\_SignalLPC2Cepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalLPC2Cepstral\\_S16\\_Adp\(3MLIB\)](#), [mllib\\_SignalLPC2Cepstral\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalLPC2Cepstral\_S16 – convert linear prediction coefficients to cepstral coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalLPC2Cepstral_S16(mllib_s16 *cepst,
 mllib_s32 cscale, const mllib_s16 *lpc, mllib_s32 lscale,
 mllib_s16 gain, mllib_s32 gscale, mllib_s32 length,
 mllib_s32 order);
```

**Description** The `mllib_SignalLPC2Cepstral_S16()` function converts linear prediction coefficients to cepstral coefficients. The user supplied scaling factor, *cscale*, will be used and the output will be saturated if necessary.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

$$c(0) = \log(G)$$

$$c(m) = a(m) + \sum_{k=1}^{m-1} c(k) * a(m-k), \quad 1 \leq m \leq M$$

$$c(m) = \sum_{k=1}^{m-1} c(k) * a(m-k), \quad m > M$$

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*cepst* The cepstral coefficients.

*cscale* The scaling factor of the cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.

*lpc* The linear prediction coefficients.

*lscale* The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.

*gain* The gain of the LPC model.

*gscale* The scaling factor of the gain of the LPC model, where `actual_data = input_data * 2**(-scaling_factor)`.

*length* The length of the cepstral coefficients.

*order*      The order of the linear prediction filter.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_SignalLPC2Cepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalLPC2Cepstral\\_S16\\_Adpt\(3MLIB\)](#), [mllib\\_SignalLPC2Cepstral\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPC2Cepstral\_S16\_Adap – convert linear prediction coefficients to cepstral coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPC2Cepstral_S16_Adap(mllib_s16 *cepst,
 mllib_s32 *cscale, const mllib_s16 *lpc, mllib_s32 lscale,
 mllib_s16 gain, mllib_s32 gscale, mllib_s32 length,
 mllib_s32 order);
```

**Description** The `mllib_SignalLPC2Cepstral_S16_Adap()` function converts linear prediction coefficients to cepstral coefficients. The scaling factor of the output data, *cscale*, will be calculated based on the actual data.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

$$c(0) = \log(G)$$

$$c(m) = a(m) + \sum_{k=1}^{m-1} \frac{k}{m} * c(k) * a(m-k), \quad 1 \leq m \leq M$$

$$c(m) = \sum_{k=1}^{m-1} \frac{k}{m} * c(k) * a(m-k), \quad m > M$$

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*cepst* The cepstral coefficients.

*cscale* The scaling factor of the cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.

*lpc* The linear prediction coefficients.

*lscale* The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.

*gain* The gain of the LPC model.

*gscale* The scaling factor of the gain of the LPC model, where `actual_data = input_data * 2**(-scaling_factor)`.

*length* The length of the cepstral coefficients.

*order*      The order of the linear prediction filter.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_SignalLPC2Cepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalLPC2Cepstral\\_S16\\_Ad\(3MLIB\)](#),  
[mllib\\_SignalLPC2Cepstral\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPC2LSP\_F32 – convert linear prediction coefficients to line spectral pair coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPC2LSP_F32(mllib_f32 *lsp, const mllib_f32 *lpc,
 mllib_s32 order);
```

**Description** The `mllib_SignalLPC2LSP_F32()` function converts linear prediction coefficients to line spectral pair coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

$$P(z) = A(z) + z^{-(M+1)} * A(z^{-1})$$

$$Q(z) = A(z) - z^{-(M+1)} * A(z^{-1})$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

Note that since  $P(z)$  is symmetric and  $Q(z)$  is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other.  $P(z)$  has a root at  $z = -1$  ( $w = \text{PI}$ ) and  $Q(z)$  has a root at  $z = 1$  ( $w = 0$ ).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

$$q = \cos(w)$$

where  $q$  is the LPS and  $w$  is the LPF.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*lsp*        The line spectral pair coefficients.

*lpc*        The linear prediction coefficients.

*order*     The order of the linear prediction filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLSP2LPC\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPC2LSP\_S16 – convert linear prediction coefficients to line spectral pair coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPC2LSP_S16(mllib_s16 *lsp, const mllib_s16 *lpc,
 mllib_s32 lscale, mllib_s32 order);
```

**Description** The `mllib_SignalLPC2LSP_S16()` function converts linear prediction coefficients to line spectral pair coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

$$P(z) = A(z) + z^{-(M+1)} * A(z^{-1})$$

$$Q(z) = A(z) - z^{-(M+1)} * A(z^{-1})$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

Note that since  $P(z)$  is symmetric and  $Q(z)$  is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other.  $P(z)$  has a root at  $z = -1$  ( $w = \pi$ ) and  $Q(z)$  has a root at  $z = 1$  ( $w = 0$ ).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

$$q = \cos(w)$$

where  $q$  is the LPS and  $w$  is the LPF.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*lsp* The line spectral pair coefficients in Q15 format.

*lpc* The linear prediction coefficients.

*lscale* The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.

*order* The order of the linear prediction filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLSP2LPC\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalLPCAutoCorrel\_F32 – perform linear predictive coding with autocorrelation method

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCAutoCorrel_F32(mllib_f32 *coeff,
 const mllib_f32 *signal, void *state);
```

**Description** The `mllib_SignalLPCAutoCorrel_F32()` function performs linear predictive coding with autocorrelation method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Bing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- coeff*      The linear prediction coefficients.
- signal*     The input signal vector.
- state*      Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCAutoCorrelInit\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetPARCOR\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

- Name** mllib\_SignalLPCAutoCorrelFree\_S16, mllib\_SignalLPCAutoCorrelFree\_F32 – clean up for autocorrelation method
- Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalLPCAutoCorrelFree_S16(void *state);
void mllib_SignalLPCAutoCorrelFree_F32(void *state);
```
- Description** Each of these functions frees the internal state structure for autocorrelation method of linear predictive coding (LPC).  
  
This function cleans up the internal state structure and releases all memory buffers.
- Parameters** Each of the functions takes the following arguments:  
  
*state*      Pointer to the internal state structure.
- Return Values** None.
- Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

- See Also** [mllib\\_SignalLPCAutoCorrelInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelInit\\_F32\(3MLIB\)](#),[mllib\\_SignalLPCAutoCorrel\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrel\\_S16\\_Adpt\(3MLIB\)](#),[mllib\\_SignalLPCAutoCorrel\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetPARCOR\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetPARCOR\\_F32\(3MLIB\)](#),[attributes\(5\)](#)

**Name** mllib\_SignalLPCAutoCorrelGetEnergy\_F32 – return the energy of the input signal

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCAutoCorrelGetEnergy_F32(
 mllib_f32 *energy, void *state);
```

**Description** The `mllib_SignalLPCAutoCorrelGetEnergy_F32()` function returns the energy of the input signal.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*energy*      The energy of the input signal.  
*state*        Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalLPCAutoCorrelInit\\_F32\(3MLIB\)](#), [mlib\\_SignalLPCAutoCorrel\\_F32\(3MLIB\)](#),  
[mlib\\_SignalLPCAutoCorrelGetPARCOR\\_F32\(3MLIB\)](#),  
[mlib\\_SignalLPCAutoCorrelFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCAutoCorrelGetEnergy\_S16,  
mllib\_SignalLPCAutoCorrelGetEnergy\_S16\_Adv – return the energy of the input signal

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalLPCAutoCorrelGetEnergy_S16(
 mllib_s16 *engery, mllib_s32 escale, void *state);

mllib_status mllib_SignalLPCAutoCorrelGetEnergy_S16_Adv(
 mllib_s16 *engery, mllib_s32 *escale, void *state);
```

**Description** Each of the functions returns the energy of the input signal.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

Note for functions with adaptive scaling (with `_Adp` postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without `_Adp` postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters** Each function takes the following arguments:

*energy*     The energy of the input signal.

*escale*     The scaling factor of the energy, where `actual_data = output_data * 2**(-scaling_factor)`.

*state*     Pointer to the internal state structure.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignallPCAutoCorrelInit\\_S16\(3MLIB\)](#), [mllib\\_SignallPCAutoCorrel\\_S16\(3MLIB\)](#), [mllib\\_SignallPCAutoCorrelGetPARCOR\\_S16\(3MLIB\)](#), [mllib\\_SignallPCAutoCorrelFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCAutoCorrelGetPARCOR\_F32 – return the partial correlation (PARCOR) coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCAutoCorrelGetPARCOR_F32(
 mllib_f32 *parcor, void *state);
```

**Description** The `mllib_SignalLPCAutoCorrelGetPARCOR_F32()` function returns the partial correlation (PARCOR) coefficients.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Bing-Hwang Juang, Prentice Hall, 1993.



**Parameters** The function takes the following arguments:

- parcor* The partial correlation (PARCOR) coefficients.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCAutoCorrelInit\\_F32\(3MLIB\)](#), [mllib\\_SignalLPCAutoCorrel\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCAutoCorrelGetPARCOR\_S16,  
mllib\_SignalLPCAutoCorrelGetPARCOR\_S16\_Adp – return the partial correlation  
(PARCOR) coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCAutoCorrelGetPARCOR_S16(
 mllib_s16 *parcor, mllib_s32 pscale, void *state);

mllib_status mllib_SignalLPCAutoCorrelGetPARCOR_S16_Adp(
 mllib_s16 *parcor, mllib_s32 *pscale, void *state);
```

**Description** Each of the functions returns the partial correlation (PARCOR) coefficients.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past M samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

Note for functions with adaptive scaling (with `_Adp` postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without `_Adp` postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

- Parameters** Each function takes the following arguments:
- parcor* The partial correlation (PARCOR) coefficients.
  - pscale* The scaling factor of the partial correlation (PARCOR) coefficients, where  $\text{actual\_data} = \text{output\_data} * 2^{**}(-\text{scaling\_factor})$ .
  - state* Pointer to the internal state structure.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCAutoCorrelInit\\_S16\(3MLIB\)](#), [mllib\\_SignalLPCAutoCorrel\\_S16\(3MLIB\)](#), [mllib\\_SignalLPCAutoCorrelGetEnergy\\_S16\(3MLIB\)](#), [mllib\\_SignalLPCAutoCorrelFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCAutoCorrelInit\_S16, mllib\_SignalLPCAutoCorrelInit\_F32 – initialization for autocorrelation method of linear predictive coding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalLPCAutoCorrelInit_S16(void *state,
 mllib_s32 length, mllib_s32 order);

mllib_status mllib_SignalLPCAutoCorrelInit_F32(void *state,
 mllib_s32 length, mllib_s32 order);
```

**Description** Each function initializes the internal state structure for autocorrelation method of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

**Parameters** Each function takes the following arguments:

- state*      Pointer to the internal state structure.
- length*    The length of the input signal vector.
- order*      The order of the linear prediction filter.

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCAutoCorrel\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetPARCOR\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLPCAutoCorrel\_S16, mlib\_SignalLPCAutoCorrel\_S16\_Adp – perform linear predictive coding with autocorrelation method

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalLPCAutoCorrel_S16(mlib_s16 *coeff,
 mlib_s32 cscale, const mlib_s16 *signal, void *state);

mlib_status mlib_SignalLPCAutoCorrel_S16_Adp(mlib_s16 *coeff,
 mlib_s32 *cscale, const mlib_s16 *signal, void *state);
```

**Description** Each function performs linear predictive coding with autocorrelation method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * r(|i-k|) = r(k), \quad k=1, \dots, M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.  $r(0)$  is the energy of the speech signal.

Note that the autocorrelation matrix  $R$  is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

Note for functions with adaptive scaling (with `_Adp` postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without `_Adp` postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters** Each function takes the following arguments:

- coeff*      The linear prediction coefficients.
- cscale*     The scaling factor of the linear prediction coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- signal*     The input signal vector with samples in Q15 format.
- state*      Pointer to the internal state structure.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCAutoCorrelInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetEnergy\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelGetPARCOR\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCAutoCorrelFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCCovariance\_F32 – perform linear predictive coding with covariance method

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCCovariance_F32(mllib_f32 *coeff,
 const mllib_f32 *signal, void *state);
```

**Description** The `mllib_SignalLPCCovariance_F32()` function performs linear predictive coding with covariance method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In covariance method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * c(i, k) = c(0, k), \quad k=1, \dots, M$$

where

$$c(i, k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k-i)$$

are the covariance coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.

Note that the covariance matrix  $R$  is a symmetric matrix, and the equations can be solved efficiently with Cholesky decomposition method.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*coeff*      The linear prediction coefficients.

*signal*     The input signal vector.

*state*      Pointer to the internal state structure.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_SignalLPCovarianceInit\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCovarianceFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalLPCCovarianceFree\_S16, mllib\_SignalLPCCovarianceFree\_F32 – clean up for covariance method

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalLPCCovarianceFree_S16(void *state);
void mllib_SignalLPCCovarianceFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for covariance method of linear predictive coding (LPC).  
  
This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:  
  
*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCCovarianceInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCCovarianceInit\\_F32\(3MLIB\)](#), [mllib\\_SignalLPCCovariance\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCCovariance\\_S16\\_Adp\(3MLIB\)](#), [mllib\\_SignalLPCCovariance\\_F32\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalLPCCovarianceInit\_S16, mllib\_SignalLPCCovarianceInit\_F32 – initialization for covariance method of linear predictive coding

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalLPCCovarianceInit_S16(void *state,
 mllib_s32 length, mllib_s32 order);

mllib_status mllib_SignalLPCCovarianceInit_F32(void *state,
 mllib_s32 length, mllib_s32 order);
```

**Description** Each function initializes the internal state structure for covariance method of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

**Parameters** Each function takes the following arguments:

*state*        Pointer to the internal state structure.

*length*      The length of the input signal vector.

*order*        The order of the linear prediction filter.

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCCovariance\\_S16\(3MLIB\)](#), [mllib\\_SignalLPCCovarianceFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCCovariance\_S16, mllib\_SignalLPCCovariance\_S16\_Adp – perform linear predictive coding with covariance method

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLPCCovariance_S16(mllib_s16 *coeff,
 mllib_s32 cscale, const mllib_s16 *signal, void *state);

mllib_status mllib_SignalLPCCovariance_S16_Adp(mllib_s16 *coeff,
 mllib_s32 *cscale, const mllib_s16 *signal, void *state);
```

**Description** Each function performs linear predictive coding with covariance method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past  $M$  samples.

$$s(n) = \sum_{i=1}^M a(i) * s(n-i) + G * u(n)$$

where  $s(*)$  is the speech signal,  $u(*)$  is the excitation signal, and  $G$  is the gain constants,  $M$  is the order of the linear prediction filter. Given  $s(*)$ , the goal is to find a set of coefficient  $a(*)$  that minimizes the prediction error  $e(*)$ .

$$e(n) = s(n) - \sum_{i=1}^M a(i) * s(n-i)$$

In covariance method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^M a(i) * c(i, k) = c(0, k), \quad k=1, \dots, M$$

where

$$c(i, k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k-i)$$

are the covariance coefficients of  $s(*)$ ,  $N$  is the length of the input speech vector.

Note that the covariance matrix  $R$  is a symmetric matrix, and the equations can be solved efficiently with Cholesky decomposition method.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

Note for functions with adaptive scaling (with `_Adp` postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without `_Adp` postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters** Each function takes the following arguments:

- coeff*      The linear prediction coefficients.
- cscale*     The scaling factor of the linear prediction coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- signal*     The input signal vector with samples in Q15 format.
- state*      Pointer to the internal state structure.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCovarianceInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCovarianceFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCPerceptWeight\_F32 – perform perceptual weighting on input signal

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalLPCPerceptWeight_F32(mllib_f32 *sigwgt,
 const mllib_f32 *signal, const mllib_f32 *lpc, mllib_f32 r1,
 mllib_f32 r2, void *state);
```

**Description** The `mllib_SignalLPCPerceptWeight_F32()` function performs perceptual weighting on input signal.

The perceptual weighting filter is defined as following.

$$W(z) = \frac{A(z*r1)}{A(z*r2)}$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

See G.723.1, G.728, G.729, G.729A, GSM EFR standards.

**Parameters** The function takes the following arguments:

*sigwgt* The weighted signal vector.

*signal* The input signal vector.

*lpc* The linear prediction coefficients.

*r1* The perceptual weighting filter coefficient, it is treated as 1 if 0 is supplied.

*r2* The perceptual weighting filter coefficient, it is treated as 1 if 0 is supplied.

*state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCPerceptWeightInit\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeightFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCPerceptWeightFree\_S16, mllib\_SignalLPCPerceptWeightFree\_F32 – clean up for perceptual weighting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_SignalLPCPerceptWeightFree_S16(void *state);
void mllib_SignalLPCPerceptWeightFree_F32(void *state);
```

**Description** Each of these functions frees the internal state structure for perceptual weighting of linear predictive coding (LPC).

This function cleans up the internal state structure and releases all memory buffers.

**Parameters** Each of the functions takes the following arguments:

*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCPerceptWeightInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeightInit\\_F32\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeight\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeight\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLPCPerceptWeightInit\_S16, mllib\_SignalLPCPerceptWeightInit\_F32 – initialization for perceptual weighting of linear predictive coding

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalLPCPerceptWeightInit_S16(void *state,
 mllib_s32 length, mllib_s32 order);

mllib_status mllib_SignalLPCPerceptWeightInit_F32(void *state,
 mllib_s32 length, mllib_s32 order);
```

**Description** Each function initializes the internal state structure for perceptual weighting of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

**Parameters** Each function takes the following arguments:

*state*      Pointer to the internal state structure.  
*length*     The length of the input signal vector.  
*order*      The order of the linear prediction filter.

**Return Values** Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPCPerceptWeight\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeightFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalLPCPerceptWeight\_S16 – perform perceptual weighting on input signal

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalLPCPerceptWeight_S16(mllib_s16 *sigwgt,
 const mllib_s16 *signal, const mllib_s16 *lpc, mllib_s32 lscale,
 mllib_s16 r1, mllib_s16 r2, void *state);
```

**Description** The `mllib_SignalLPCPerceptWeight_S16()` function performs perceptual weighting on input signal.

The perceptual weighting filter is defined as following.

$$W(z) = \frac{A(z*r1)}{A(z*r2)}$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

See G.723.1, G.728, G.729, G.729A, GSM EFR standards.

**Parameters** The function takes the following arguments:

- sigwgt* The weighted signal vector, the signal samples are in Q15 format.
- signal* The input signal vector, the signal samples are in Q15 format.
- lpc* The linear prediction coefficients.
- lscale* The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.
- r1* The perceptual weighting filter coefficient, the coefficient is in Q15 format, it is treated as 1 if 0 is supplied.
- r2* The perceptual weighting filter coefficient, the coefficient is in Q15 format, it is treated as 1 if 0 is supplied.
- state* Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mllib\\_SignalLPCPerceptWeightInit\\_S16\(3MLIB\)](#),  
[mllib\\_SignalLPCPerceptWeightFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLPCPitchAnalyze\_F32 – perform open-loop pitch analysis

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalLPCPitchAnalyze_F32(mlib_s32 *pitch,
 const mlib_f32 *sigwgt, const mlib_s32 *region,
 mlib_s32 length);
```

**Description** The `mlib_SignalLPCPitchAnalyze_F32()` function performs open-loop pitch analysis.

The open-loop pitch analysis uses perceptual weighted signal and is done with following steps.

In the first step, three maxima of the correlation

$$R(k) = \sum_{j=0}^{N-1} sw(j) * sw(j-k)$$

where  $N = \text{length}$ , is located for each of the three search regions.

In the second step, the retained maxima  $R(T_i)$ ,  $i=0, 1, 2$  are normalized as following.

$$R_n(T_i) = \frac{R(T_i)}{\sqrt{\sum_{j=0}^{N-1} sw(j-T_i)^2}}, \quad i=0, 1, 2$$

where  $N = \text{length}$ .

In the third step, the best open-loop delay  $T_{opt}$  is determined as following.

```
Topt = T0
if (Rn(t1) ≥ (0.85 * Rn(Topt)))
 Topt = t1
if (Rn(t2) ≥ (0.85 * Rn(Topt)))
 Topt = t2
```

See G.729, G.729A, GSM EFR standards.

**Parameters** The function takes the following arguments:

- |               |                                                                                                                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pitch</i>  | The speech pitch estimated.                                                                                                                                                                                                                                         |
| <i>sigwgt</i> | The weighted signal vector. <i>sigwgt</i> points to the current sample of the weighted signal vector, <i>length</i> samples must be available after this point, and $\text{MAX}\{\text{region}[i], i=0, 1, \dots, 5\}$ samples must be available before this point. |
| <i>region</i> | The lower/upper boundaries of the three search regions, where <i>region</i> [2*i] is the lower boundary of search region <i>i</i> and <i>region</i> [2*i+1] is the upper boundary of search region <i>i</i> .                                                       |

*length*     The length of the signal vectors over which the correlation is calculated.

**Return Values**   The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**   See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**   [mllib\\_SignalLPCPitchAnalyze\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalLPCPitchAnalyze\_S16 – perform open-loop pitch analysis

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalLPCPitchAnalyze_S16(mlib_s32 *pitch,
 const mlib_s16 *sigwgt, const mlib_s32 *region,
 mlib_s32 length);
```

**Description** The `mlib_SignalLPCPitchAnalyze_S16()` function performs open-loop pitch analysis. The open-loop pitch analysis uses perceptual weighted signal and is done with following steps.

In the first step, three maxima of the correlation

$$R(k) = \sum_{j=0}^{N-1} sw(j) * sw(j-k)$$

where  $N = \text{length}$ , is located for each of the three search regions.

In the second step, the retained maxima  $R(T_i)$ ,  $i=0, 1, 2$  are normalized as following.

$$Rn(ti) = \frac{R(T_i)}{\sqrt{\sum_{j=0}^{N-1} sw(j-T_i)^2}}, \quad i=0, 1, 2$$

where  $N = \text{length}$ .

In the third step, the best open-loop delay  $T_{opt}$  is determined as following.

```
Topt = T0
if (Rn(t1) ≥ (0.85 * Rn(Topt)))
 Topt = t1
if (Rn(t2) ≥ (0.85 * Rn(Topt)))
 Topt = t2
```

See G.729, G.729A, GSM EFR standards.

**Parameters** The function takes the following arguments:

|               |                                                                                                                                                                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pitch</i>  | The speech pitch estimated.                                                                                                                                                                                                                                                                    |
| <i>sigwgt</i> | The weighted signal vector with samples in Q15 format. <i>sigwgt</i> points to the current sample of the weighted signal vector, <i>length</i> samples must be available after this point, and $\text{MAX}\{\text{region}[i], i=0, 1, \dots, 5\}$ samples must be available before this point. |
| <i>region</i> | The lower/upper boundaries of the three search regions, where <i>region</i> [2*i] is the lower boundary of search region <i>i</i> and <i>region</i> [2*i+1] is the upper boundary of search region <i>i</i> .                                                                                  |

*length*      The length of the signal vectors over which the correlation is calculated.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_SignalLPCPitchAnalyze\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalLSP2LPC\_F32 – convert line spectral pair coefficients to linear prediction coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalLSP2LPC_F32(mllib_f32 *lpc,
 const mllib_f32 *lsp, mllib_s32 order);
```

**Description** The `mllib_SignalLSP2LPC_F32()` function converts line spectral pair coefficients to linear prediction coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

$$P(z) = A(z) + z^{-(M+1)} * A(z^{-1})$$

$$Q(z) = A(z) - z^{-(M+1)} * A(z^{-1})$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

Note that since  $P(z)$  is symmetric and  $Q(z)$  is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other.  $P(z)$  has a root at  $z = -1$  ( $w = \text{PI}$ ) and  $Q(z)$  has a root at  $z = 1$  ( $w = 0$ ).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

$$q = \cos(w)$$

where  $q$  is the LPS and  $w$  is the LPF.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

*lpc*        The linear prediction coefficients.

*lsp*        The line spectral pair coefficients.

*order*     The order of the linear prediction filter.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPC2LSP\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalLSP2LPC\_S16, mllib\_SignalLSP2LPC\_S16\_Adp – convert line spectral pair coefficients to linear prediction coefficients

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalLSP2LPC_S16(mllib_s16 *lpc,
 mllib_s32 lscale, const mllib_s16 *lsp, mllib_s32 order);

mllib_status mllib_SignalLSP2LPC_S16_Adp(mllib_s16 *lpc,
 mllib_s32 *lscale, const mllib_s16 *lsp, mllib_s32 order);
```

**Description** Each of the functions in this group converts line spectral pair coefficients to linear prediction coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

$$P(z) = A(z) + z^{-(M+1)} * A(z^{-1})$$

$$Q(z) = A(z) - z^{-(M+1)} * A(z^{-1})$$

where  $A(z)$  is the inverse filter

$$A(z) = 1 - \sum_{i=1}^M a(i) * z^{-i}$$

Note that since  $P(z)$  is symmetric and  $Q(z)$  is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other.  $P(z)$  has a root at  $z = -1$  ( $w = \pi$ ) and  $Q(z)$  has a root at  $z = 1$  ( $w = 0$ ).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

$$q = \cos(w)$$

where  $q$  is the LPS and  $w$  is the LPF.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

Note for functions with adaptive scaling (with `_Adp` postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without `_Adp` postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters** Each function takes the following arguments:

- lpc*        The linear prediction coefficients.
- lscale*    The scaling factor of the line spectral pair coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- lsp*        The line spectral pair coefficients in Q15 format.
- order*      The order of the linear prediction filter.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLPC2LSP\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMelCepstral\_F32 – perform cepstral analysis in mel frequency scale

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMelCepstral_F32(mllib_f32 *cepst,
 const mllib_f32 *signal, void *state);
```

**Description** The `mllib_SignalMelCepstral_F32()` function performs cepstral analysis in mel frequency scale.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.

```

 +-----+ +-----+
 | Linear | | Inverse |
... ----->| to |----->| Fourier |----->
 X'(k) | Mel Scale | X''(m) | Transform | c(n)
 +-----+ +-----+

```

where  $X'(k)$  is defined in linear frequency scale and  $X''(m)$  is defined in mel frequency scale.

The mel frequency scale is defined as following.

$$\text{freq\_mel} = \text{melmul} * \text{LOG10}(1 + \text{freq\_linear} / \text{meldiv})$$

where `freq_mel` is the frequency in mel scale, `freq_linear` is the frequency in linear scale, `melmul` is the multiplying factor, `meldiv` is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

```
0 f1 f2 f3 fp fp+1 fp+2 fp+3 fp+q
|---|---|---| ... |---|----|-----| ... | ... -> freq
```

where `fp = melbgn`, `fp+q = melend`, `p = nlinear`, `q = nmel`; the filters number 1 to `p` are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number `p+1` to `p+q` are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- cepst*      The cepstral coefficients.
- signal*     The input signal vector.
- state*      Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMelCepstralInit\\_F32\(3MLIB\)](#), [mllib\\_SignalMelCepstralFree\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMelCepstralFree\_S16, mllib\_SignalMelCepstralFree\_F32 – clean up for cepstral analysis in mel frequency scale

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`void mllib_SignalMelCepstralFree_S16(void *state);`

`void mllib_SignalMelCepstralFree_F32(void *state);`

**Description** Each of these functions frees the internal *state* structure and releases all memory buffers for cepstral analysis in mel frequency scale.

**Parameters** Each of the functions takes the following arguments:

*state*      Pointer to the internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMelCepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalMelCepstral\\_F32\(3MLIB\)](#), [mllib\\_SignalMelCepstral\\_S16\\_Adpt\(3MLIB\)](#), [mllib\\_SignalMelCepstralInit\\_S16\(3MLIB\)](#), [mllib\\_SignalMelCepstralInit\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMelCepstralInit\_S16, mllib\_SignalMelCepstralInit\_F32 – initialization for cepstral analysis in mel frequency scale

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalMelCepstralInit_S16(void *state, mllib_s32 nlinear,
 mllib_s32 nmel, mllib_f32 melbgn, mllib_f32 melend, mllib_f32 meldiv,
 mllib_s32 order);

mllib_status mllib_SignalMelCepstralInit_F32(void *state, mllib_s32 nlinear,
 mllib_s32 nmel, mllib_f32 melbgn, mllib_f32 melend, mllib_f32 meldiv,
 mllib_s32 order);
```

**Description** Each of these functions initializes the internal state structure for cepstral analysis in mel frequency scale.

The init function performs internal state structure allocation and global initialization. Per function call initialization is done in each function, so the same internal state structure can be reused for multiple function calls.

**Parameters** Each of the functions takes the following arguments:

- state* Pointer to the internal state structure.
- nlinear* The number of band pass filters in linear frequency scale.
- nmel* The number of band pass filters in mel frequency scale.
- melbgn* The begin radian frequency of the mel scale filter bank defined in linear frequency scale, where  $0 \leq \text{melbgn} < \text{melend} \leq \text{PI}$ , *melbgn* is ignored if *nlinear* = 0.
- melend* The end radian frequency of the mel scale filter bank defined in linear frequency scale, where  $0 \leq \text{melbgn} < \text{melend} \leq \text{PI}$ .
- meldiv* The dividing factor in linear to mel scale conversion, linear scale is measured in radians, with *PI* corresponding to half the sampling rate.
- order* The order of the input signal vector and the cepstral coefficients, where  $\text{length} = 2 * \text{order}$ .

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_SignalMelCepstral_S16(3MLIB)`, `mlib_SignalMelCepstral_F32(3MLIB)`,  
`mlib_SignalMelCepstral_S16_Adp(3MLIB)`, `mlib_SignalMelCepstralFree_S16(3MLIB)`,  
`mlib_SignalMelCepstralFree_F32(3MLIB)`, `attributes(5)`

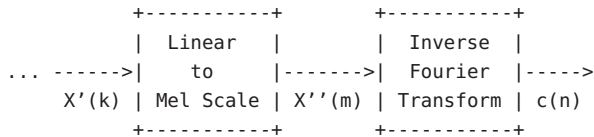
**Name** mlib\_SignalMelCepstral\_S16 – perform cepstral analysis in mel frequency scale

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_SignalMelCepstral_S16(mlib_s16 *cepst,
 mlib_s32 cscale, const mlib_s16 *signal, void *state);
```

**Description** The `mlib_SignalMelCepstral_S16()` function performs cepstral analysis in mel frequency scale. The user supplied scaling factor will be used and the output will be saturated if necessary.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.



where  $X'(k)$  is defined in linear frequency scale and  $X''(m)$  is defined in mel frequency scale.

The mel frequency scale is defined as following.

```
freq_mel = melmul * LOG10(1 + freq_linear / meldiv)
```

where `freq_mel` is the frequency in mel scale, `freq_linear` is the frequency in linear scale, `melmul` is the multiplying factor, `meldiv` is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

```
0 f1 f2 f3 fp fp+1 fp+2 fp+3 fp+q
|---|---|---| ... |---|----|-----| ... | ... -> freq
```

where `fp = melbgn`, `fp+q = melend`, `p = nlinear`, `q = nmel`; the filters number 1 to `p` are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number `p+1` to `p+q` are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.



**Parameters** The function takes the following arguments:

- cepst* The cepstral coefficients.
- cscale* The scaling factor of cepstral coefficients, where  $\text{actual\_data} = \text{output\_data} * 2^{**}(-\text{scaling\_factor})$ .
- signal* The input signal vector, the signal samples are in Q15 format.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMelCepstralInit\\_S16\(3MLIB\)](#), [mllib\\_SignalMelCepstral\\_S16\\_Adp\(3MLIB\)](#), [mllib\\_SignalMelCepstralFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

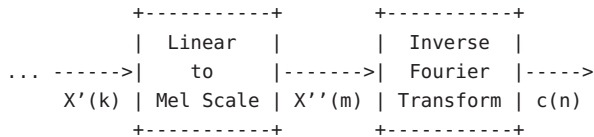
**Name** mllib\_SignalMelCepstral\_S16\_AdP – perform cepstral analysis in mel frequency scale

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMelCepstral_S16_AdP(mllib_s16 *cepst,
 mllib_s32 *cscale, const mllib_s16 *signal, void *state);
```

**Description** The `mllib_SignalMelCepstral_S16_AdP()` function performs cepstral analysis in mel frequency scale. The scaling factor of the output data will be calculated based on the actual data.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.



where  $X'(k)$  is defined in linear frequency scale and  $X''(m)$  is defined in mel frequency scale.

The mel frequency scale is defined as following.

$$\text{freq\_mel} = \text{melmul} * \text{LOG10}(1 + \text{freq\_linear} / \text{meldiv})$$

where `freq_mel` is the frequency in mel scale, `freq_linear` is the frequency in linear scale, `melmul` is the multiplying factor, `meldiv` is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

```
0 f1 f2 f3 fp fp+1 fp+2 fp+3 fp+q
|---|---|---| ... |---|----|-----| ... | ... -> freq
```

where `fp` = `melbgn`, `fp+q` = `melend`, `p` = `nlinear`, `q` = `nmel`; the filters number 1 to `p` are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number `p+1` to `p+q` are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.

See *Digital Signal Processing* by Alan V. Oppenheim and Ronald W. Schaffer, Prentice Hall, 1974.

See *Fundamentals of Speech Recognition* by Lawrence Rabiner and Biing-Hwang Juang, Prentice Hall, 1993.

**Parameters** The function takes the following arguments:

- cepst* The cepstral coefficients.
- cscale* The scaling factor of cepstral coefficients, where  $\text{actual\_data} = \text{output\_data} * 2^{**}(-\text{scaling\_factor})$ .
- signal* The input signal vector, the signal samples are in Q15 format.
- state* Pointer to the internal state structure.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMelCepstralInit\\_S16\(3MLIB\)](#), [mllib\\_SignalMelCepstral\\_S16\(3MLIB\)](#), [mllib\\_SignalMelCepstralFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMerge\_F32S\_F32 – merge

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMerge_F32S_F32(mllib_f32 *dst, const mllib_f32 *ch0,
 const mllib_f32 *ch1, mllib_s32 n);
```

**Description** The `mllib_SignalMerge_F32S_F32()` function merges two signal arrays to form a stereo signal array.

**Parameters** The function takes the following arguments:

- dst*     Output stereo signal array. `dst[2*i]` contains Channel 0, and `dst[2*i+1]` contains Channel 1.
- ch0*     Input signal array of Channel 0.
- ch1*     Input signal array of Channel 1.
- n*        Number of samples in the source signal arrays.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMerge\\_S16S\\_S16\(3MLIB\)](#), [mllib\\_SignalSplit\\_F32\\_F32S\(3MLIB\)](#),  
[mllib\\_SignalSplit\\_S16\\_S16S\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMerge\_S16S\_S16 – merge

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMerge_S16S_S16(mllib_s16 *dst, const mllib_s16 *ch0,
 const mllib_s16 *ch1, mllib_s32 n);
```

**Description** The `mllib_SignalMerge_S16S_S16()` function merges two signal arrays to form a stereo signal array.

**Parameters** The function takes the following arguments:

- dst*     Output stereo signal array. `dst[2*i]` contains Channel 0, and `dst[2*i+1]` contains Channel 1.
- ch0*     Input signal array of Channel 0.
- ch1*     Input signal array of Channel 1.
- n*        Number of samples in the source signal arrays.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMerge\\_F32S\\_F32\(3MLIB\)](#), [mllib\\_SignalSplit\\_F32\\_F32S\(3MLIB\)](#),  
[mllib\\_SignalSplit\\_S16\\_S16S\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBartlett\_F32, mllib\_SignalMulBartlett\_F32S – Bartlett windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulBartlett_F32(mllib_f32 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulBartlett_F32S(mllib_f32 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*     Input and output signal array.

*n*           Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulBartlett\_F32\_F32, mlib\_SignalMulBartlett\_F32S\_F32S – Bartlett windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulBartlett_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 n);

mlib_status mlib_SignalMulBartlett_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*dst*     Output signal array.  
*src*     Input signal array.  
*n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBartlett\_S16, mllib\_SignalMulBartlett\_S16S – Bartlett windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMulBartlett_S16(mllib_s16 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulBartlett_S16S(mllib_s16 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*     Input and output signal array.  
*n*           Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalMulBartlett\_S16\_S16, mlib\_SignalMulBartlett\_S16S\_S16S – Bartlett windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulBartlett_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 n);

mlib_status mlib_SignalMulBartlett_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*dst*     Output signal array.  
*src*     Input signal array.  
*n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBlackman\_F32, mllib\_SignalMulBlackman\_F32S – Blackman windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_SignalMulBlackman_F32(mllib_f32 *srcdst, mllib_f32 alpha,
 mllib_s32 n);

mllib_status mllib_SignalMulBlackman_F32S(mllib_f32 *srcdst, mllib_f32 alpha,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- alpha*     Blackman window parameter.  $-1 < \alpha < 0$ .
- n*          Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBlackman\_F32\_F32, mllib\_SignalMulBlackman\_F32S\_F32S – Blackman windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulBlackman_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, mllib_f32 alpha, mllib_s32 n);

mllib_status mllib_SignalMulBlackman_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, mllib_f32 alpha, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*dst*        Output signal array.

*src*        Input signal array.

*alpha*      Blackman window parameter.  $-1 < \alpha < 0$ .

*n*          Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBlackman\_S16, mllib\_SignalMulBlackman\_S16S – Blackman windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulBlackman_S16(mllib_s16 *srcdst, mllib_f32 alpha,
 mllib_s32 n);

mllib_status mllib_SignalMulBlackman_S16S(mllib_s16 *srcdst, mllib_f32 alpha,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- alpha*     Blackman window parameter.  $-1 < \alpha < 0$ .
- n*          Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulBlackman\_S16\_S16, mllib\_SignalMulBlackman\_S16S\_S16S – Blackman windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulBlackman_S16_S16(mllib_s16 *dst,
 const mllib_s16 *src, mllib_f32 alpha, mllib_s32 n);

mllib_status mllib_SignalMulBlackman_S16S_S16S(mllib_s16 *dst,
 const mllib_s16 *src, mllib_f32 alpha, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Bartlett window.

**Parameters** Each of the functions takes the following arguments:

*dst*        Output signal array.

*src*        Input signal array.

*alpha*      Blackman window parameter.  $-1 < \alpha < 0$ .

*n*          Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMul\_F32, mllib\_SignalMul\_F32S – multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMul_F32(mllib_f32 *src1dst,
 const mllib_f32 *src2, mllib_s32 n);

mllib_status mllib_SignalMul_F32S(mllib_f32 *src1dst,
 const mllib_f32 *src2, mllib_s32 n);
```

**Description** Each of these functions performs multiplication.

**Parameters** Each of the functions takes the following arguments:

*src1dst*     The first input and the output signal array.

*src2*        The second input signal array.

*n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalMul\_F32\_F32, mllib\_SignalMul\_F32S\_F32S – multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMul_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src1, const mllib_f32 *src2, mllib_s32 n);

mllib_status mllib_SignalMul_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src1, const mllib_f32 *src2, mllib_s32 n);
```

**Description** Each of these functions performs multiplication.

**Parameters** Each of the functions takes the following arguments:

- src1dst*     The output signal array.
- src1*        The first input signal array
- src2*        The second input signal array.
- n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalMulHamming\_F32, mllib\_SignalMulHamming\_F32S – Bartlett windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulHamming_F32(mllib_f32 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulHamming_F32S(mllib_f32 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*     Input and output signal array.

*n*           Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalMulHamming\_F32\_F32, mllib\_SignalMulHamming\_F32S\_F32S – Hamming windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulHamming_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, mllib_s32 n);

mllib_status mllib_SignalMulHamming_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

- dst*     Output signal array.
- src*     Input signal array.
- n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulHamming\_S16, mllib\_SignalMulHamming\_S16S – Bartlett windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulHamming_S16(mllib_s16 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulHamming_S16S(mllib_s16 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*     Input and output signal array.

*n*           Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulHamming\_S16\_S16, mllib\_SignalMulHamming\_S16S\_S16S – Hamming windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_SignalMulHamming_S16_S16(mllib_s16 *dst,  
const mllib_s16 *src, mllib_s32 n);  
  
mllib_status mllib_SignalMulHamming_S16S_S16S(mllib_s16 *dst,  
const mllib_s16 *src, mllib_s32 n);`

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

- dst*     Output signal array.
- src*     Input signal array.
- n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulHanning\_F32, mllib\_SignalMulHanning\_F32S – Hanning windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulHanning_F32(mllib_f32 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulHanning_F32S(mllib_f32 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hanning window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*      Source and destination signal array.

*n*            Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulHanning\_F32\_F32, mlib\_SignalMulHanning\_F32S\_F32S – Hanning windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalMulHanning_F32_F32(mlib_f32 *dst,
 mlib_s32const mlib_f32 *src, n);

mlib_status mlib_SignalMulHanning_F32S_F32S(mlib_f32 *dst,
 mlib_s32const mlib_f32 *src, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hanning window.

**Parameters** Each of the functions takes the following arguments:

*dst* Destination signal array.  
*src* Source signal array.  
*n* Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulHanning\_S16, mllib\_SignalMulHanning\_S16S – Hanning windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulHanning_S16(mllib_s16 *srcdst, mllib_s32 n);
mllib_status mllib_SignalMulHanning_S16S(mllib_s16 *srcdst, mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hanning window.

**Parameters** Each of the functions takes the following arguments:

*srcdst*      Source and destination signal array.

*n*            Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulHanning\_S16\_S16, mlib\_SignalMulHanning\_S16S\_S16S – Hanning windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulHanning_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 n);

mlib_status mlib_SignalMulHanning_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hanning window.

**Parameters** Each of the functions takes the following arguments:

*dst* Destination signal array.  
*src* Source signal array.  
*n* Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalMulKaiser\_F32, mllib\_SignalMulKaiser\_F32S – Kaiser windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_SignalMulKaiser_F32(mllib_f32 *srcdst, mllib_f32 beta,  
                                        mllib_s32 n);  
  
mllib_status mllib_SignalMulKaiser_F32S(mllib_f32 *srcdst, mllib_f32 beta,  
                                        mllib_s32 n);`

**Description** Each of these functions performs multiplication of the Kaiser window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*      Source and destination signal array.
- beta*        Kaiser window parameter.
- n*            Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalMulKaiser\_F32\_F32, mlib\_SignalMulKaiser\_F32S\_F32S – Kaiser windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalMulKaiser_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, mlib_f32 beta, mlib_s32 n);

mlib_status mlib_SignalMulKaiser_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, mlib_f32 beta, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Kaiser window.

**Parameters** Each of the functions takes the following arguments:

*dst*      Output signal array.  
*src*      Input signal array.  
*beta*     Kaiser window parameter.  
*n*        Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulKaiser\_S16, mllib\_SignalMulKaiser\_S16S – Kaiser windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulKaiser_S16(mllib_s16 *srcdst, mllib_f32 beta,
 mllib_s32 n);

mllib_status mllib_SignalMulKaiser_S16S(mllib_s16 *srcdst, mllib_f32 beta,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the Kaiser window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*      Source and destination signal array.
- beta*        Kaiser window parameter.
- n*            Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulKaiser\_S16\_S16, mlib\_SignalMulKaiser\_S16S\_S16S – Kaiser windowing multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalMulKaiser_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, mlib_f32 beta, mlib_s32 n);

mlib_status mlib_SignalMulKaiser_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, mlib_f32 beta, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Kaiser window.

**Parameters** Each of the functions takes the following arguments:

*dst*      Output signal array.  
*src*      Input signal array.  
*beta*     Kaiser window parameter.  
*n*        Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mlib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulRectangular\_F32, mllib\_SignalMulRectangular\_F32S – rectangular windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulRectangular_F32(mllib_f32 *srcdst, mllib_s32 m,
 mllib_s32 n);

mllib_status mllib_SignalMulRectangular_F32S(mllib_f32 *srcdst, mllib_s32 m,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the rectangular window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- m*         Rectangular window parameter.
- n*         Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulRectangular\\_F32\\_F32\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_F32\(3MLIB\)](#),  
[mllib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulRectangular\_F32\_F32, mlib\_SignalMulRectangular\_F32S\_F32S – rectangular windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulRectangular_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 m, mlib_s32 n);

mlib_status mlib_SignalMulRectangular_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

*dst*     Output signal array.  
*src*     Input signal array.  
*m*       Rectangular window parameter.  
*n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulBlackman\\_F32\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulHamming\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulHanning\\_F32\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulKaiser\\_F32\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_F32\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulRectangular\\_F32\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_F32\(3MLIB\)](#),  
[mlib\\_SignalMulWindow\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulRectangular\_S16, mllib\_SignalMulRectangular\_S16S – rectangular windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulRectangular_S16(mllib_s16 *srcdst, mllib_s32 m,
 mllib_s32 n);

mllib_status mllib_SignalMulRectangular_S16S(mllib_s16 *srcdst, mllib_s32 m,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication of the rectangular window.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- m*         Rectangular window parameter.
- n*         Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#),  
[mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulRectangular\_S16\_S16, mlib\_SignalMulRectangular\_S16S\_S16S – rectangular windowing multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulRectangular_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 m, mlib_s32 n);

mlib_status mlib_SignalMulRectangular_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions performs multiplication of the Hamming window.

**Parameters** Each of the functions takes the following arguments:

*dst*     Output signal array.  
*src*     Input signal array.  
*m*       Rectangular window parameter.  
*n*       Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBartlett\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulBlackman\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHamming\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulHanning\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mlib\\_SignalMulKaiser\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\(3MLIB\)](#),  
[mlib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [mlib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_SignalMul\_S16\_S16\_Sat, mllib\_SignalMul\_S16S\_S16S\_Sat – multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMul_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, mllib_s32 n);

mllib_status mllib_SignalMul_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, mllib_s32 n);
```

**Description** Each of these functions performs multiplication.

**Parameters** Each of the functions takes the following arguments:

- dst*      Output signal array.
- src1*     The first input signal array.
- src2*     The second input signal array.
- n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMul\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalMul\_S16\_Sat, mllib\_SignalMul\_S16S\_Sat – multiplication

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMul_S16_Sat(mllib_s16 *src1dst,
 const mllib_s16 *src2, mllib_s32 n);

mllib_status mllib_SignalMul_S16S_Sat(mllib_s16 *src1dst,
 const mllib_s16 *src2, mllib_s32 n);
```

**Description** Each of these functions performs multiplication.

**Parameters** Each of the functions takes the following arguments:

*src1dst*     The first input and the output signal array.  
*src2*        The second input signal array.  
*n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMul\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSAdd\_F32, mllib\_SignalMulSAdd\_F32S – multiplication by a scalar plus addition

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulSAdd_F32(mllib_f32 *src1dst,
 const mllib_f32 *const mllib_f32 *src2, c, mllib_s32 n);

mllib_status mllib_SignalMulSAdd_F32S(mllib_f32 *src1dst,
 const mllib_f32 *const mllib_f32 *src2, const mllib_f32 *c,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar plus addition.

**Parameters** Each of the functions takes the following arguments:

*src1dst*     The first input and the output signal array.

*src2*        The second input signal array.

*c*            Scaling factor. In the stereo version, *c*[0] contains the scaling factor for channel 0, and *c*[1] holds the scaling factor for channel 1.

*n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSAdd\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalMulSAdd\_F32\_F32, mlib\_SignalMulSAdd\_F32S\_F32S – multiplication by a scalar plus addition

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulSAdd_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src1, const mlib_f32 *src2, const mlib_f32 *c,
 mlib_s32 n);

mlib_status mlib_SignalMulSAdd_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src1, const mlib_f32 *src2, const mlib_f32 *c,
 mlib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

*dst*      Output signal array.

*src1*     The first input signal array.

*src2*     The second input signal array.

*c*        Scaling factor. In the stereo version, *c*[0] contains the scaling factor for channel 0, and *c*[1] holds the scaling factor for channel 1.

*n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mlib\\_SignalMulSAdd\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSAdd\_S16\_S16\_Sat, mllib\_SignalMulSAdd\_S16S\_S16S\_Sat – multiplication by a scalar plus addition

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulSAdd_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_SignalMulSAdd_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, const mllib_s16 *c,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

- dst*      Output signal array.
- src1*     The first input signal array.
- src2*     The second input signal array.
- c*        Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSAdd\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSAdd\_S16\_Sat, mllib\_SignalMulSAdd\_S16S\_Sat – multiplication by a scalar plus addition

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_SignalMulSAdd_S16_Sat(mllib_s16 *src1dst,  
const mllib_s16 *src2, const mllib_s16 *c, mllib_s32 n);  
  
mllib_status mllib_SignalMulSAdd_S16S_Sat(mllib_s16 *src1dst,  
const mllib_s16 *src2, const mllib_s16 *c, mllib_s32 n);`

**Description** Each of these functions performs multiplication by a scalar plus addition.

**Parameters** Each of the functions takes the following arguments:

- src1dst* The first input and the output signal array.
- src2* The second input signal array.
- c* Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- n* Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSAdd\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulS\_F32, mllib\_SignalMulS\_F32S – multiplication by a scalar

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulS_F32(mllib_f32 *srcdst, const mllib_f32 *c,
 mllib_s32 n);

mllib_status mllib_SignalMulS_F32S(mllib_f32 *srcdst, const mllib_f32 *c,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- c*             Scaling factor. In the stereo version, *c*[0] contains the scaling factor for channel 0, and *c*[1] holds the scaling factor for channel 1.
- n*             Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulS\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulS\_F32\_F32, mllib\_SignalMulS\_F32S\_F32S – multiplication by a scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_SignalMulS_F32_F32(mllib_f32 *dst, const mllib_f32 *src,  
const mllib_f32 *c, mllib_s32 n);  
  
mllib_status mllib_SignalMulS_F32S_F32S(mllib_f32 *dst, const mllib_f32 *src,  
const mllib_f32 *c, mllib_s32 n);`

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

- dst*     Output signal array.
- src*     Input signal array.
- c*        Scaling factor. In the stereo version, *c*[0] contains the scaling factor for channel 0, and *c*[1] holds the scaling factor for channel 1.
- n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulS\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulShift\_S16\_S16\_Sat, mllib\_SignalMulShift\_S16S\_S16S\_Sat – multiplication with shifting

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulShift_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2,
 mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalMulShift_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2,
 mllib_s32 shift, mllib_s32 n);
```

**Description** Each of these functions performs multiplication with shifting.

**Parameters** Each of the functions takes the following arguments:

- dst*        Output signal array.
- src1*      The first input signal array.
- src2*      The second input signal array.
- shift*     Left shifting factor.
- n*         Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulShift\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_SignalMulShift\_S16\_Sat, mlib\_SignalMulShift\_S16S\_Sat – multiplication with shifting

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalMulShift_S16_Sat(mlib_s16 *src1dst,
 const mlib_s16 *src2, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalMulShift_S16S_Sat(mlib_s16 *src1dst,
 const mlib_s16 *src2, mlib_s32 shift, mlib_s32 n);
```

**Description** Each of these functions performs multiplication with shifting.

**Parameters** Each of the functions takes the following arguments:

*src1dst*     The first input and the output signal array.

*src2*        The second input signal array.

*shift*       Left shifting factor.

*n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulShift\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulS\_S16\_S16\_Sat, mllib\_SignalMulS\_S16S\_S16S\_Sat – multiplication by a scalar

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulS_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_SignalMulS_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *c, mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

- dst*     Output signal array.
- src*     Input signal array.
- c*        Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulS\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulS\_S16\_Sat, mllib\_SignalMulS\_S16S\_Sat – multiplication by a scalar

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulS_S16_Sat(mllib_s16 *srcdst, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_SignalMulS_S16S_Sat(mllib_s16 *srcdst, const mllib_s16 *c,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- c*             Scaling factor. The scaling factor is in Q15 format. In the stereo version; *c*[0] contains the scaling factor for channel 0, and *c*[1] holds the scaling factor for channel 1.
- n*             Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulS\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSShiftAdd\_S16\_S16\_Sat, mllib\_SignalMulSShiftAdd\_S16S\_S16S\_Sat – multiplication by a scalar plus addition

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulSShiftAdd_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, const mllib_s16 *c,
 mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalMulSShiftAdd_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src1, const mllib_s16 *src2, const mllib_s16 *c,
 mllib_s32 shift, mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar with shifting plus addition.

**Parameters** Each of the functions takes the following arguments:

- dst*      Output signal array.
- src1*     The first input signal array.
- src2*     The second input signal array.
- c*        Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- shift*    Left shifting factor.
- n*        Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSShiftAdd\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** `mlib_SignalMulSShiftAdd_S16_Sat`, `mlib_SignalMulSShiftAdd_S16S_Sat` – multiplication by a scalar plus addition

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalMulSShiftAdd_S16_Sat(mlib_s16 *src1dst,
 const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 shift,
 mlib_s32 n);

mlib_status mlib_SignalMulSShiftAdd_S16S_Sat(mlib_s16 *src1dst,
 const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 shift,
 mlib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar with shifting plus addition.

**Parameters** Each of the functions takes the following arguments:

*src1dst*     The first input and the output signal array.

*src2*        The second input signal array.

*c*            Scaling factor. The scaling factor is in Q15 format. In the stereo version; `c[0]` contains the scaling factor for channel 0, and `c[1]` holds the scaling factor for channel 1.

*n*            Number of samples in the input signal arrays.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_SignalMulSShiftAdd\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSShift\_S16\_S16\_Sat, mllib\_SignalMulSShift\_S16S\_S16S\_Sat – multiplication by a scalar with shifting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulSShift_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *c, mllib_s32 shift,
 mllib_s32 n);

mllib_status mllib_SignalMulSShift_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *c, mllib_s32 shift,
 mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar with shifting.

**Parameters** Each of the functions takes the following arguments:

- dst* Destination signal array.
- src* Source signal array.
- c* Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- shift* Left shifting factor.
- n* Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSShift\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulSShift\_S16\_Sat, mllib\_SignalMulSShift\_S16S\_Sat – multiplication by a scalar with shifting

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulSShift_S16_Sat(mllib_s16 *srcdst,
 const mllib_s16 *c, mllib_s32 shift, mllib_s32 n);

mllib_status mllib_SignalMulSShift_S16S_Sat(mllib_s16 *srcdst,
 const mllib_s16 *c, mllib_s32 shift, mllib_s32 n);
```

**Description** Each of these functions performs multiplication by a scalar with shifting.

**Parameters** Each of the functions takes the following arguments:

*srcdst*     Source and destination signal array.

*c*             Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.

*shift*        Left shifting factor.

*n*             Number of samples in the input signal arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMulSShift\\_S16\\_S16\\_Sat\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalMulWindow\_F32 – windowing

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalMulWindow_F32(mllib_f32 *srcdst,
 const mllib_f32 *window, mllib_s32 n);
```

**Description** The `mllib_SignalMulWindow_F32()` function performs a windowing operation.

**Parameters** The function takes the following arguments:

- srcdst*     Input and output signal array.
- window*    Window coefficient array. The window coefficients are in Q15 format.
- n*           Number of samples in signal and window arrays.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mlib\_SignalMulWindow\_F32\_F32 – windowing

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalMulWindow_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *window, mlib_s32 n);
```

**Description** The `mlib_SignalMulWindow_F32_F32()` function performs a windowing operation.

**Parameters** The function takes the following arguments:

*dst*            Output signal array.  
*src*            Input signal array.  
*window*        Window coefficient array. The window coefficients are in Q15 format.  
*n*              Number of samples in signal and window arrays.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalMulWindow\_F32S – windowing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalMulWindow_F32S(mllib_f32 *srcdst,
 const mllib_f32 *window, mllib_s32 n);
```

**Description** The mllib\_SignalMulWindow\_F32S() function performs a windowing operation.

**Parameters** The function takes the following arguments:

- srcdst*     Input and output signal array are in stereo format where srcdst[0] contains the values for channel 0, and srcdst[1] holds the valuesfor channel 1.
- window*     Window coefficient array. The window coefficients are in Q15 format.
- n*           Number of samples in signal and window arrays.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalMulWindow\_F32S\_F32S – windowing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalMulWindow_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, const mllib_f32 *window, mllib_s32 n);
```

**Description** The `mllib_SignalMulWindow_F32S_F32S()` function performs a windowing operation.

**Parameters** The function takes the following arguments:

- dst* Output signal array is in stereo format where `dst[0]` contains the values for channel 0, and `dst[1]` holds the valuesfor channel 1.
- src* Input signal array is in stereo format where `src[0]` contains the values for channel 0, and `src[1]` holds the valuesfor channel 1.
- window* Window coefficient array. The window coefficients are in Q15 format.
- n* Number of samples in signal and window arrays.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalMulWindow\_S16, mllib\_SignalMulWindow\_S16S – windowing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulWindow_S16(mllib_s16 *srcdst,
 const mllib_s16 *window, mllib_s32 n);

mllib_status mllib_SignalMulWindow_S16S(mllib_s16 *srcdst,
 const mllib_s16 *window, mllib_s32 n);
```

**Description** Each of these functions performs a windowing operation.

**Parameters** Each of the functions takes the following arguments:

- srcdst*     Input and output signal array.
- window*    Window coefficient array. The window coefficients are in Q15 format.
- n*           Number of samples in signal and window arrays.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also** [mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name**

mllib\_SignalMulWindow\_S16\_S16, mllib\_SignalMulWindow\_S16S\_S16S – windowing

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalMulWindow_S16_S16(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 n);

mllib_status mllib_SignalMulWindow_S16S_S16S(mllib_s16 *dst,
 const mllib_s16 *src, const mllib_s16 *window, mllib_s32 n);
```

**Description**

Each of these functions performs a windowing operation.

**Parameters**

Each of the functions takes the following arguments:

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>dst</i>    | Output signal array.                                                 |
| <i>src</i>    | Input signal array.                                                  |
| <i>window</i> | Window coefficient array. The window coefficients are in Q15 format. |
| <i>n</i>      | Number of samples in signal and window arrays.                       |

**Return Values**

Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTEVALUE |
|---------------------|----------------|
| Interface Stability | Committed      |
| MT-Level            | MT-Safe        |

**See Also**

[mllib\\_SignalMulBartlett\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBartlett\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\(3MLIB\)](#), [mllib\\_SignalMulBlackman\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHamming\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\(3MLIB\)](#), [mllib\\_SignalMulHanning\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\(3MLIB\)](#), [mllib\\_SignalMulKaiser\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\(3MLIB\)](#), [mllib\\_SignalMulRectangular\\_S16\\_S16\(3MLIB\)](#), [mllib\\_SignalMulWindow\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalNLMSFilter, mlib\_SignalNLMSFilterInit\_S16\_S16, mlib\_SignalNLMSFilterInit\_S16S\_S16S, mlib\_SignalNLMSFilterInit\_F32\_F32, mlib\_SignalNLMSFilterInit\_F32S\_F32S, mlib\_SignalNLMSFilter\_S16\_S16\_Sat, mlib\_SignalNLMSFilter\_S16S\_S16S\_Sat, mlib\_SignalNLMSFilter\_F32\_F32, mlib\_SignalNLMSFilter\_F32S\_F32S, mlib\_SignalNLMSFilterNonAdapt\_S16\_S16\_Sat, mlib\_SignalNLMSFilterNonAdapt\_S16S\_S16S\_Sat, mlib\_SignalNLMSFilterNonAdapt\_F32\_F32, mlib\_SignalNLMSFilterNonAdapt\_F32S\_F32S, mlib\_SignalNLMSFilterFree\_S16\_S16, mlib\_SignalNLMSFilterFree\_S16S\_S16S, mlib\_SignalNLMSFilterFree\_F32\_F32, mlib\_SignalNLMSFilterFree\_F32S\_F32S – normalized least mean square (NLMS) adaptive filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_SignalNLMSFilterInit_S16_S16(void **filter,
 const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_S16S_S16S(void **filter,
 const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_F32_F32(void **filter,
 const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_F32S_F32S(void **filter,
 const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilter_S16_S16_Sat(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_S16S_S16S_Sat(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_S16_S16_Sat(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_S16S_S16S_Sat(mlib_s16 *dst,
 const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

void mlib_SignalNLMSFilterFree_S16_S16(void *filter);

void mlib_SignalNLMSFilterFree_S16S_S16S(void *filter);

void mlib_SignalNLMSFilterFree_F32_F32(void *filter);
```

```
void mllib_SignalNLMSFilterFree_F32S_F32S(void *filter);
```

**Description** The normalized LMS adaptive algorithm is summarized as follows:

1. Initialize the weights  $W_k(i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ .
2. Initialize previous source elements  $X_o(i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ .
3. Read  $X_k(t)$  from *src* and  $Y_k(t)$  from *ref*,  $t = 0, 1, \dots, n - 1$ .
4. Compute filter output:  $n_k = \sum(W_k(i) * X_k(t - i))$ ,  $i = 0, 1, \dots, \text{tap} - 1$ . If  $i > t$ , use previous source elements stored in the *Xo* vector.
5. Compute source elements power:  $P_{wk} = \sum(X_k(t - i) * X_k(t - i))$ ,  $i = 0, 1, \dots, \text{tap} - 1$ . If  $i > t$ , use previous source elements stored in the *Xo* vector.
6. Store filter output :  $dst[t] = n_k$ .
7. Compute the error estimate:  $E_k = Y_k - n_k$ .
8. Compute factor  $BE0 = 2 * \text{beta} * E_k / P_{wk}$ .
9. Update filter weights:  $W_k(i) += BE0 * X_k(t - i)$ ,  $i = 0, 1, \dots, \text{tap} - 1$ . If  $i > t$ , use previous source elements stored in the *Xo* vector.
10. Next  $t$ , go to step 3.
11. Store  $N$  ending source elements in previous source elements vector *Xo*: if  $N > n$ ,  $N = n$ ; else  $N = \text{tap}$ .

Each of the `FilterInit` functions allocates memory for the internal filter structure and converts the parameters into the internal representation.

Each of the `Filter` functions applies the NLMS adaptive filter on one signal packet and updates the filter states.

Each of the `FilterNoAdapt` functions applies the NLMS filter on one signal packet and updates the filter states but without changing the filter weights.

Each of the `FilterFree` functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes some of the following arguments:

- |               |                                                 |
|---------------|-------------------------------------------------|
| <i>filter</i> | Internal filter structure.                      |
| <i>flt</i>    | Filter coefficient array.                       |
| <i>tap</i>    | Taps of the filter.                             |
| <i>beta</i>   | Error weighting factor. $0 < \text{beta} < 1$ . |
| <i>dst</i>    | Destination signal array.                       |
| <i>src</i>    | Source signal array.                            |
| <i>ref</i>    | Reference or “desired” signal array.            |

*n*            Number of samples in the source signal array.

**Return Values** Each of the `FilterInit`, `Filter` and `FilterNonAdapt` functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. The `FilterFree` functions don't return anything.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalLMSFilter\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignalQuant2\_S16\_F32 – float to 16-bit quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalQuant2_S16_F32(mllib_s16 *dst,
 const mllib_f32 *src, const mllib_f32 thresh, mllib_s32 length,
 mllib_s16 offset, mllib_s32 n);
```

**Description** The `mllib_SignalQuant2_S16_F32()` function quantizes a signal array by using the following equation:

```
X = x(n) n = 0, 1, ...
Z = z(n) n = 0, 1, ...
= offset for x(n) < t(0)
= offset + k for t(k) ≤ x(n) < t(k+1)
= offset + length - 1 for x(n) ≥ t(length - 1)
```

**Parameters** The function takes the following arguments:

*dst*            Output signal array

*src*            Input signal array .

*thresh*        Array of thresholds.

*length*        Length of the array of thresholds.

*offset*        Offset for thresholds.

*n*             Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalQuant2\_S16S\_F32S – float to 16-bit quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalQuant2_S16S_F32S(mllib_s16 *dst,
 const mllib_f32 *src, const mllib_f32 thresh, mllib_s32 length,
 mllib_s16 offset, mllib_s32 n);
```

**Description** The mllib\_SignalQuant2\_S16S\_F32S() function quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) & n &= 0, 1, \dots \\ Z &= z(n) & n &= 0, 1, \dots \\ &= \text{offset} & \text{for } x(n) < t(0) \\ &= \text{offset} + k & \text{for } t(k) \leq x(n) < t(k+1) \\ &= \text{offset} + \text{length} - 1 & \text{for } x(n) \geq t(\text{length} - 1) \end{aligned}$$

**Parameters** The function takes the following arguments:

- dst*

Output signal array in two-channel interleaved stereo format.
- src*

Input signal array in two-channel interleaved stereo format.
- thresh*

Array of thresholds.
- length*

Length of the array of thresholds.
- offset*

Offset for thresholds.
- n*

Number of samples in the input signal array.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalQuant\_S16\_F32 – float to 16-bit quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalQuant_S16_F32(mllib_s16 *dst,
 const mllib_f32 *src, const mllib_f32 *thresh, mllib_s32 n);
```

**Description** The `mllib_SignalQuant_S16_F32()` function quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) & n &= 0, 1, \dots \\ Z &= z(n) & n &= 0, 1, \dots \\ &= -32768 & \text{for } x(n) < t(-32768) \\ &= k & \text{for } t(k) \leq x(n) < t(k+1) \\ &= +32767 & \text{for } x(n) \geq t(+32767) \end{aligned}$$

**Parameters** The function takes the following arguments:

*dst*            Output signal array

*src*            Input signal array .

*thresh*        Array of 65536 thresholds.

*n*              Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalQuant\_S16S\_F32S – float to 16-bit quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalQuant_S16S_F32S(mllib_s16 *dst,
 const mllib_f32 *src, const mllib_f32 *thresh, mllib_s32 n);
```

**Description** The mllib\_SignalQuant\_S16S\_F32S() function quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) & n &= 0, 1, \dots \\ Z &= z(n) & n &= 0, 1, \dots \\ &= -32768 & \text{for } x(n) < t(-32768) \\ &= k & \text{for } t(k) \leq x(n) < t(k+1) \\ &= +32767 & \text{for } x(n) \geq t(+32767) \end{aligned}$$

**Parameters** The function takes the following arguments:

- dst* Output signal array in two-channel interleaved stereo format.
- src* Input signal array in two-channel interleaved stereo format.
- thresh* Array of 65536 thresholds.
- n* Number of samples in the input signal array.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalQuant\_U8\_F32 – float to 8-bit quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalQuant_U8_F32(mllib_u8 *dst,
 const mllib_f32 *src, const mllib_f32 *thresh, mllib_s32 n);
```

**Description** The `mllib_SignalQuant_U8_F32()` function quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) \quad n = 0, 1, \dots \\ Z &= z(n) \quad n = 0, 1, \dots \\ &= 0 \quad \text{for } x(n) < t(0) \\ &= k \quad \text{for } t(k) \leq x(n) < t(k+1) \\ &= 255 \quad \text{for } x(n) \geq t(255) \end{aligned}$$

**Parameters** The function takes the following arguments:

- dst*        Output signal array
- src*        Input signal array .
- thresh*     Array of 256 thresholds.
- n*         Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalQuant\_U8\_S16, mllib\_SignalQuant\_U8S\_S16S – 16-bit to 8-bit quantization

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_SignalQuant_U8_S16(mllib_u8 *dst,  
                                  const mllib_s16 *src, const mllib_s16 *thresh, mllib_s32 n);  
  
mllib_status mllib_SignalQuant_U8S_S16S(mllib_u8 *dst,  
                                  const mllib_s16 *src, const mllib_s16 *thresh, mllib_s32 n);`

**Description** Each of these functions quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) \quad n = 0, 1, \dots \\ Z &= z(n) \quad n = 0, 1, \dots \\ &= 0 \quad \text{for } x(n) < t(0) \\ &= k \quad \text{for } t(k) \leq x(n) < t(k+1) \\ &= 255 \quad \text{for } x(n) \geq t(255) \end{aligned}$$

**Parameters** Each of the functions takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- thresh*    Array of 256 thresholds.
- n*         Number of samples in the input signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mlib\_SignalQuant\_U8S\_F32S – float to 8-bit quantization

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalQuant_U8S_F32S(mlib_u8 *dst,
 const mlib_f32 *src, const mlib_f32 *thresh, mlib_s32 n);
```

**Description** The `mlib_SignalQuant_U8S_F32S()` function quantizes a signal array by using the following equation:

$$\begin{aligned} X &= x(n) & n &= 0, 1, \dots \\ Z &= z(n) & n &= 0, 1, \dots \\ &= 0 & \text{for } x(n) < t(0) \\ &= k & \text{for } t(k) \leq x(n) < t(k+1) \\ &= 255 & \text{for } x(n) \geq t(255) \end{aligned}$$

**Parameters** The function takes the following arguments:

*dst* Output signal array in two-channel interleaved stereo format.

*src* Input signal array in two-channel interleaved stereo format.

*thresh* Array of 256 thresholds.

*n* Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIR\_F32\_F32 – resampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalReSampleFIR_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalReSampleFIR_F32_F32()` function performs rational sample rate conversion with FIR filtering between the upsampling and downsampling.

**Parameters** The function takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- state*      Internal state structure.
- n*          Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalReSampleFIR\_F32S\_F32S – resampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalReSampleFIR_F32S_F32S(mllib_f32 *dst,
 const mllib_f32 *src, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalReSampleFIR_F32S_F32S()` function performs rational sample rate conversion with FIR filtering between the upsampling and downsampling.

**Parameters** The function takes the following arguments:

*dst* Output signal array in two-channel interleaved stereo format.

*src* Input signal array in two-channel interleaved stereo format.

*state* Internal state structure.

*n* Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIRFree\_F32\_F32 – resampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalReSampleFIRFree_F32_F32(void *state);
```

**Description** The `mllib_SignalReSampleFIRFree_F32_F32()` function releases the memory allocated for the internal state structure for rational sample rate conversion with FIR filtering between upsampling and downsampling.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIRFree\_F32S\_F32S – resampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalReSampleFIRFree_F32S_F32S(void *state);
```

**Description** The `mllib_SignalReSampleFIRFree_F32S_F32S()` function releases the memory allocated for the internal state structure for rational sample rate conversion with FIR filtering between upsampling and downsampling.

**Parameters** The function takes the following arguments:

*state* Internal state structure.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIRFree\_S16\_S16, mllib\_SignalReSampleFIRFree\_S16S\_S16S – resampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalReSampleFIRFree_S16_S16(void *state);
void mllib_SignalReSampleFIRFree_S16S_S16S(void *state);
```

**Description** Each of these functions releases the memory allocated for the internal state structure for rational sample rate conversion with FIR filtering between upsampling and downsampling.

**Parameters** Each of the functions takes the following arguments:

*state*     Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalReSampleFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalReSampleFIRInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIRInit\_S16\_S16, mllib\_SignalReSampleFIRInit\_S16S\_S16S, mllib\_SignalReSampleFIRInit\_F32\_F32, mllib\_SignalReSampleFIRInit\_F32S\_F32S – initialization for resampling with filtering

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_SignalReSampleFIRInit_S16_S16(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor,
 mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);

mllib_status mllib_SignalReSampleFIRInit_S16S_S16S(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor,
 mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);

mllib_status mllib_SignalReSampleFIRInit_F32_F32(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor,
 mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);

mllib_status mllib_SignalReSampleFIRInit_F32S_F32S(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor,
 mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);
```

**Description** Each of these functions allocates memory for the internal state structure and converts the parameters into an internal representation for rational sample rate conversion with FIR filtering between upsampling and downsampling.

**Parameters** Each of the functions takes the following arguments:

*state* Internal state structure.

*flt* Filter coefficient array, two-channel interleaved in the cases of stereo.

*tap* Taps of the filter.

*ufactor* Factor by which to upsample.

*uphase* Phase in upsampling.  $0 \leq \text{uphase} < \text{ufactor}$ .

*dfactor* Factor by which to downsample.

*dphase* Phase in downsampling.  $0 \leq \text{dphase} < \text{dfactor}$ .

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalReSampleFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalReSampleFIR\\_F32\\_F32\(3MLIB\)](#),  
[mllib\\_SignalReSampleFIRFree\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalReSampleFIRFree\\_F32\\_F32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalReSampleFIR\_S16\_S16\_Sat, mllib\_SignalReSampleFIR\_S16S\_S16S\_Sat – resampling with filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalReSampleFIR_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *state, mllib_s32 n);

mllib_status mllib_SignalReSampleFIR_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *state, mllib_s32 n);
```

**Description** Each of these functions performs rational sample rate conversion with FIR filtering between the upsampling and downsampling.

**Parameters** Each of the functions takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- state*      Internal state structure.
- n*          Number of samples in the input signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalSineWave\_F32 – sine wave generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalSineWave_F32(mllib_f32 *sine, void *state,
 mllib_s32 n);
```

**Description** The `mllib_SignalSineWave_F32()` function generates one packet of sine wave and updates the internal state.

**Parameters** The function takes the following arguments:

- sine*      Generated sine wave array.
- state*      Internal state structure.
- n*          Length of the generated sine wave array in number of samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalSineWaveFree\_F32 – sine wave generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`  
  
`void mllib_SignalSineWaveFree_F32(void *state);`

**Description** The `mllib_SignalSineWaveFree_F32()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalSineWave\\_S16\(3MLIB\)](#), [mllib\\_SignalSineWaveInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalSineWaveFree\_S16 – sine wave generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalSineWaveFree_S16(void *state);
```

**Description** The `mllib_SignalSineWaveFree_S16()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalSineWaveInit\_F32 – sine wave generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalSineWaveInit_F32(void **state,
 mllib_f32 mag, mllib_f32 freq, mllib_f32 phase);
```

**Description** The `mllib_SignalSineWaveInit_F32()` function allocates memory for an internal state structure and converts the parameters of the wave to an internal representation.

**Parameters** The function takes the following arguments:

*state* Internal state structure.

*mag* Magnitude of sine wave to be generated, in Q15 format.

*freq* Angular frequency of the sine wave to be generated, measured in radians per sample.

*phase* Start phase of the sine wave to be generated, measured in radians.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalSineWaveInit\_S16 – sine wave generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalSineWaveInit_S16(void **state,
 mllib_s16 mag, mllib_f32 freq, mllib_f32 phase);
```

**Description** The `mllib_SignalSineWaveInit_S16()` function allocates memory for an internal state structure and converts the parameters of the wave to an internal representation.

**Parameters** The function takes the following arguments:

- state* Internal state structure.
- mag* Magnitude of sine wave to be generated, in Q15 format.
- freq* Angular frequency of the sine wave to be generated, measured in radians per sample.
- phase* Start phase of the sine wave to be generated, measured in radians.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalSineWave\\_S16\(3MLIB\)](#), [mllib\\_SignalSineWaveFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalSineWave\_S16 – sine wave generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalSineWave_S16(mllib_s16 *sine, void *state,
 mllib_s32 n);
```

**Description** The `mllib_SignalSineWave_S16()` function generates one packet of sine wave and updates the internal state.

**Parameters** The function takes the following arguments:

- sine*      Generated sine wave array.
- state*     Internal state structure.
- n*         Length of the generated sine wave array in number of samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalSineWaveFree\\_S16\(3MLIB\)](#), [mllib\\_SignalSineWaveInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalSplit\_F32\_F32S – split

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalSplit_F32_F32S(mllib_f32 *ch0,
 mllib_f32 *ch1, const mllib_f32 *src, mllib_s32 n);
```

**Description** The following function splits a stereo signal array into two signal arrays.

**Parameters** The function takes the following arguments:

- ch0* Destination signal array of Channel 0.
- ch1* Destination signal array of Channel 1.
- src* Source stereo signal array. *src*[2\*i] contains Channel 0, and *src*[2\*i+1] contains Channel 1.
- n* Number of samples in the source signal array.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalSplit\_S16\_S16S – split

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`mllib_status mllib_SignalSplit_S16_S16S(mllib_s16 *ch0,  
mllib_s16 *ch1, const mllib_s16 *src, mllib_s32 n);`

**Description** The following function splits a stereo signal array into two signal arrays.

**Parameters** The function takes the following arguments:

*ch0* Destination signal array of Channel 0.

*ch1* Destination signal array of Channel 1.

*src* Source stereo signal array. *src*[2\**i*] contains Channel 0, and *src*[2\**i*+1] contains Channel 1.

*n* Number of samples in the source signal array.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalMerge\\_S16S\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignaluLaw2ALaw – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignaluLaw2ALaw(mllib_u8 *acode,
 const mllib_u8 *ucode, mllib_s32 n);
```

**Description** The `mllib_SignaluLaw2ALaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (formerly CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

- acode*     A-law code array.
- ucode*     m-law code array.
- n*         Number of samples in the input array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalALaw2Linear\(3MLIB\)](#), [mllib\\_SignalALaw2uLaw\(3MLIB\)](#),  
[mllib\\_SignalLinear2ALaw\(3MLIB\)](#), [mllib\\_SignalLinear2uLaw\(3MLIB\)](#),  
[mllib\\_SignaluLaw2Linear\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_SignaluLaw2Linear – ITU G.711 m-law and A-law compression and decompression

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignaluLaw2Linear(mllib_s16 *pcm,
 const mllib_u8 *ucode, mllib_s32 n);
```

**Description** The `mllib_SignaluLaw2Linear()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (formerly CCITT) G.711 specification.

**Parameters** The function takes the following arguments:

*pcm*        Linear PCM sample array.  
*ucode*      m-law code array.  
*n*            Number of samples in the input array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalALaw2Linear\(3MLIB\)](#), [mllib\\_SignalALaw2uLaw\(3MLIB\)](#),  
[mllib\\_SignalLinear2ALaw\(3MLIB\)](#), [mllib\\_SignalLinear2uLaw\(3MLIB\)](#),  
[mllib\\_SignaluLaw2ALaw\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIR\_F32\_F32 – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalUpSampleFIR_F32_F32(mllib_f32 *dst,
 const mllib_f32 *src, void *state, mllib_s32 n);
```

**Description** The `mllib_SignalUpSampleFIR_F32_F32()` function performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state.

**Parameters** The function takes the following arguments:

*dst*        Output signal array.

*src*        Input signal array.

*state*      Internal state structure.

*n*          Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mlib\_SignalUpSampleFIR\_F32S\_F32S – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_SignalUpSampleFIR_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, void *state, mlib_s32 n);
```

**Description** The `mlib_SignalUpSampleFIR_F32S_F32S()` function performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state.

**Parameters** The function takes the following arguments:

*dst*        Output stereo signal array. `src[2*i]` contains Channel 0, and `src[2*i+1]` contains Channel 1.

*src*        Source stereo signal array. `src[2*i]` contains Channel 0, and `src[2*i+1]` contains Channel 1.

*state*      Internal state structure.

*n*          Number of samples in the input signal array.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIRFree\_F32\_F32 – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalUpSampleFIRFree_F32_F32(void *state);
```

**Description** The `mllib_SignalUpSampleFIRFree_F32_F32()` function releases the memory allocated for the internal state structure for upsampling immediately followed by FIR filtering.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIRFree\_F32S\_F32S – upsampling with filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

`void mllib_SignalUpSampleFIRFree_F32S_F32S(void *state);`

**Description** The `mllib_SignalUpSampleFIRFree_F32S_F32S()` function releases the memory allocated for the internal state structure for upsampling immediately followed by FIR filtering.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIRFree\_S16\_S16, mllib\_SignalUpSampleFIRFree\_S16S\_S16S – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalUpSampleFIRFree_S16_S16(void *state);
void mllib_SignalUpSampleFIRFree_S16S_S16S(void *state);
```

**Description** Each of these functions releases the memory allocated for the internal state structure for upsampling immediately followed by FIR filtering.

**Parameters** Each of the functions takes the following arguments:

*state*     Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalUpSampleFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalUpSampleFIRInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIRInit\_F32\_F32 – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalUpSampleFIRInit_F32_F32(void **state,
 const mllib_f32 *flt, mllib_s32 tap, mllib_s32 factor,
 mllib_s32 phase);
```

**Description** The `mllib_SignalUpSampleFIRInit_F32_F32()` function allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

**Parameters** The function takes the following arguments:

*state* Internal state structure.

*flt* Filter coefficient array.

*tap* Taps of the filter.

*factor* Factor by which to upsample.

*phase* Parameter that determines the relative position of an input value, within the output signal.  $0 \leq \text{phase} < \text{factor}$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIRInit\_F32S\_F32S – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalUpSampleFIRInit_F32S_F32S(void **state,
 const mllib_f32 *flt, mllib_s32 tap, mllib_s32 factor,
 mllib_s32 phase);
```

**Description** The mllib\_SignalUpSampleFIRInit\_F32S\_F32S() function allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

**Parameters** The function takes the following arguments:

|               |                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>state</i>  | Internal state structure.                                                                                                            |
| <i>flt</i>    | Filter coefficient array in two-channel stereo format. src[2*i] contains channel 0, and src[2*i+1] contains channel 1 array.         |
| <i>tap</i>    | Taps of the filter.                                                                                                                  |
| <i>factor</i> | Factor by which to upsample.                                                                                                         |
| <i>phase</i>  | Parameter that determines the relative position of an input value, within the output signal. $0 \leq \text{phase} < \text{factor}$ . |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalUpSampleFIRInit\_S16\_S16, mllib\_SignalUpSampleFIRInit\_S16S\_S16S – upsampling with filtering

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_SignalUpSampleFIRInit_S16_S16(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 factor,
 mlib_s32 phase);
```

```
mllib_status mllib_SignalUpSampleFIRInit_S16S_S16S(void **state,
 const mlib_f32 *flt, mlib_s32 tap, mlib_s32 factor,
 mlib_s32 phase);
```

**Description** Each of these functions allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

**Parameters** Each of the functions takes the following arguments:

*state* Internal state structure.

*flt* Filter coefficient array.

*tap* Taps of the filter.

*factor* Factor by which to upsample.

*phase* Parameter that determines the relative position of an input value, within the output signal.  $0 \leq \text{phase} < \text{factor}$ .

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalUpSampleFIR\\_S16\\_S16\\_Sat\(3MLIB\)](#),  
[mllib\\_SignalUpSampleFIRFree\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalUpSampleFIR\_S16\_S16\_Sat, mllib\_SignalUpSampleFIR\_S16S\_S16S\_Sat – upsampling with filtering

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalUpSampleFIR_S16_S16_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *state, mllib_s32 n);

mllib_status mllib_SignalUpSampleFIR_S16S_S16S_Sat(mllib_s16 *dst,
 const mllib_s16 *src, void *state, mllib_s32 n);
```

**Description** Each of these functions performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state.

**Parameters** Each of the functions takes the following arguments:

- dst*        Output signal array.
- src*        Input signal array.
- state*      Internal state structure.
- n*          Number of samples in the input signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalUpSampleFIRFree\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_SignalUpSampleFIRInit\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_SignalUpSample\_S16\_S16, mlib\_SignalUpSample\_S16S\_S16S, mlib\_SignalUpSample\_F32\_F32, mlib\_SignalUpSample\_F32S\_F32S – signal upsampling

**Synopsis** cc [ *flag...* ] *file...* -lmlib [ *library...* ]  
#include <mlib.h>

```
mlib_status mlib_SignalUpSample_S16_S16(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 factor, mlib_s32 phase,
 mlib_s32 n);

mlib_status mlib_SignalUpSample_S16S_S16S(mlib_s16 *dst,
 const mlib_s16 *src, mlib_s32 factor, mlib_s32 phase,
 mlib_s32 n);

mlib_status mlib_SignalUpSample_F32_F32(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 factor, mlib_s32 phase,
 mlib_s32 n);

mlib_status mlib_SignalUpSample_F32S_F32S(mlib_f32 *dst,
 const mlib_f32 *src, mlib_s32 factor, mlib_s32 phase,
 mlib_s32 n);
```

**Description** Each of these functions performs upsampling.

For monaural signals, the following equation is used:

$$\begin{aligned} \text{dst}[i] &= \text{src}[k] & \text{if } i == k * \text{factor} + \text{phase} \\ \text{dst}[i] &= 0 & \text{if } i \neq k * \text{factor} + \text{phase} \end{aligned}$$

where  $k = 0, 1, \dots, (n - 1)$ ;  $i = 0, 1, \dots, (n * \text{factor} - 1)$ .

For stereo signals, the following equation is used:

$$\begin{aligned} \text{dst}[2*i] &= \text{src}[2*k] & \text{if } i == k * \text{factor} + \text{phase} \\ \text{dst}[2*i] &= 0 & \text{if } i \neq k * \text{factor} + \text{phase} \end{aligned}$$

$$\begin{aligned} \text{dst}[2*i + 1] &= \text{src}[2*k + 1] & \text{if } i == k * \text{factor} + \text{phase} \\ \text{dst}[2*i + 1] &= 0 & \text{if } i \neq k * \text{factor} + \text{phase} \end{aligned}$$

where  $k = 0, 1, \dots, (n - 1)$ ;  $i = 0, 1, \dots, (n * \text{factor} - 1)$ .

**Parameters** Each of the functions takes the following arguments:

*dst*      Output signal array.

*src*      Input signal array.

*factor*    Factor by which to upsample.  $\text{factor} \geq 1$ .

*phase*    Parameter that determines relative position of an input value, within the output signal.  $0 \leq \text{phase} < \text{factor}$ .

*n*        Number of samples in the input signal array.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalDownSample\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalWhiteNoise\_F32 – white noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalWhiteNoise_F32(mllib_f32 *wnoise,
void *state, mllib_s32 n);
```

**Description** The `mllib_SignalWhiteNoise_F32()` function generates one packet of white noise and updates the internal state.

**Parameters** The function takes the following arguments:

*wnoise*      Generated white noise array.

*state*        Internal state structure.

*n*            Length of the generated sine wave array in number of samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalWhiteNoiseFree\_F32 – white noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_SignalWhiteNoiseFree_F32(void *state);
```

**Description** The `mllib_SignalWhiteNoiseFree_F32()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
  
*state* Internal state structure.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_SignalWhiteNoiseFree\_S16 – white noise generation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`  
  
`void mllib_SignalWhiteNoiseFree_S16(void *state);`

**Description** The `mllib_SignalWhiteNoiseFree_S16()` function releases the memory allocated for the internal state's structure.

**Parameters** The function takes the following arguments:  
  
*state*      Internal state structure.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalWhiteNoise\\_S16\(3MLIB\)](#), [mllib\\_SignalWhiteNoiseInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalWhiteNoiseInit\_F32 – white noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_SignalWhiteNoiseInit_F32(void **state, mllib_f32 mag,
 mllib_f32 seed);
```

**Description** The `mllib_SignalWhiteNoiseInit_F32()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

**Parameters** The function takes the following arguments:

- state* Internal state structure.
- mag* Magnitude of white noise to be generated, in Q15 format.
- seed* Seed value for the pseudorandom number generator.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_SignalWhiteNoiseInit\_S16 – white noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalWhiteNoiseInit_S16(void **state,
 mllib_s16 mag, mllib_s16 seed);
```

**Description** The `mllib_SignalWhiteNoiseInit_S16()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

**Parameters** The function takes the following arguments:

*state* Internal state structure.

*mag* Magnitude of white noise to be generated, in Q15 format.

*seed* Seed value for the pseudorandom number generator.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalWhiteNoise\\_S16\(3MLIB\)](#), [mllib\\_SignalWhiteNoiseFree\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_SignalWhiteNoise\_S16 – white noise generation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_SignalWhiteNoise_S16(mllib_s16 *wnoise,
void *state, mllib_s32 n);
```

**Description** The `mllib_SignalWhiteNoise_S16()` function generates one packet of white noise and updates the internal state.

**Parameters** The function takes the following arguments:

- wnoise*      Generated white noise array.
- state*       Internal state structure.
- n*           Length of the generated sine wave array in number of samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_SignalWhiteNoiseFree\\_S16\(3MLIB\)](#), [mllib\\_SignalWhiteNoiseInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorAddS\_U8\_Mod, mlib\_VectorAddS\_U8\_Sat, mlib\_VectorAddS\_U8C\_Mod, mlib\_VectorAddS\_U8C\_Sat, mlib\_VectorAddS\_S8\_Mod, mlib\_VectorAddS\_S8\_Sat, mlib\_VectorAddS\_S8C\_Mod, mlib\_VectorAddS\_S8C\_Sat, mlib\_VectorAddS\_S16\_Mod, mlib\_VectorAddS\_S16\_Sat, mlib\_VectorAddS\_S16C\_Mod, mlib\_VectorAddS\_S16C\_Sat, mlib\_VectorAddS\_S32\_Mod, mlib\_VectorAddS\_S32\_Sat, mlib\_VectorAddS\_S32C\_Mod, mlib\_VectorAddS\_S32C\_Sat – vector addition to scalar, in place

**Synopsis** cc [ *flag*... ] *file*... -lmllib [ *library*... ]  
#include <mlib.h>

```

mlib_status mlib_VectorAddS_U8_Mod(mlib_u8 *xz,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_U8_Sat(mlib_u8 *xz,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_U8C_Mod(mlib_u8 *xz,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_U8C_Sat(mlib_u8 *xz,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S8_Mod(mlib_s8 *xz,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S8_Sat(mlib_s8 *xz,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S8C_Mod(mlib_s8 *xz,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S8C_Sat(mlib_s8 *xz,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16_Mod(mlib_s16 *xz,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16_Sat(mlib_s16 *xz,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_Mod(mlib_s16 *xz,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_Sat(mlib_s16 *xz,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_Mod(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_Sat(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32C_Mod(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 n);

```

```
mllib_status mllib_VectorAddS_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);
```

**Description** Each of these functions performs an in-place addition of a scalar to a vector.

For real data, the following equation is used:

$$xz[i] = c[0] + xz[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} xz[2*i] &= c[0] + xz[2*i] \\ xz[2*i + 1] &= c[1] + xz[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the source and destination vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorAddS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorAddS\_U8\_U8\_Mod, mllib\_VectorAddS\_U8\_U8\_Sat, mllib\_VectorAddS\_U8C\_U8C\_Mod, mllib\_VectorAddS\_U8C\_U8C\_Sat, mllib\_VectorAddS\_S8\_S8\_Mod, mllib\_VectorAddS\_S8\_S8\_Sat, mllib\_VectorAddS\_S8C\_S8C\_Mod, mllib\_VectorAddS\_S8C\_S8C\_Sat, mllib\_VectorAddS\_S16\_U8\_Mod, mllib\_VectorAddS\_S16\_U8\_Sat, mllib\_VectorAddS\_S16\_S8\_Mod, mllib\_VectorAddS\_S16\_S8\_Sat, mllib\_VectorAddS\_S16\_S16\_Mod, mllib\_VectorAddS\_S16\_S16\_Sat, mllib\_VectorAddS\_S16C\_U8C\_Mod, mllib\_VectorAddS\_S16C\_U8C\_Sat, mllib\_VectorAddS\_S16C\_S8C\_Mod, mllib\_VectorAddS\_S16C\_S8C\_Sat, mllib\_VectorAddS\_S16C\_S16C\_Mod, mllib\_VectorAddS\_S16C\_S16C\_Sat, mllib\_VectorAddS\_S32\_S16\_Mod, mllib\_VectorAddS\_S32\_S16\_Sat, mllib\_VectorAddS\_S32\_S32\_Mod, mllib\_VectorAddS\_S32\_S32\_Sat, mllib\_VectorAddS\_S32C\_S16C\_Mod, mllib\_VectorAddS\_S32C\_S16C\_Sat, mllib\_VectorAddS\_S32C\_S32C\_Mod, mllib\_VectorAddS\_S32C\_S32C\_Sat – vector addition to scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorAddS_U8_U8_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_U8_U8_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_U8C_U8C_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_U8C_U8C_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S8_S8_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S8_S8_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S8C_S8C_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S16_U8_Mod(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S16_U8_Sat(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorAddS_S16_S8_Mod(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);
```

```
mlib_status mlib_VectorAddS_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_U8C_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_U8C_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_S8C_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_S8C_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_S16_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_S16_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_S32_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32_S32_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32C_S16C_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32C_S16C_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32C_S32C_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorAddS_S32C_S32C_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
```

**Description** Each of these functions adds a scalar to a vector.

For real data, the following equation is used:

$$z[i] = c[0] + x[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= c[0] + x[2*i] \\ z[2*i + 1] &= c[1] + x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the first element of the destination vector.
- $x$  Pointer to the first element of the source vector.
- $c$  Pointer to the source scalar. When the function is used with complex data types,  $c[0]$  contains the scalar for the real part, and  $c[1]$  contains the scalar for the imaginary part.
- $n$  Number of elements in the vectors.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorAddS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorAdd\_U8\_Mod, mlib\_VectorAdd\_U8\_Sat, mlib\_VectorAdd\_U8C\_Mod, mlib\_VectorAdd\_U8C\_Sat, mlib\_VectorAdd\_S8\_Mod, mlib\_VectorAdd\_S8\_Sat, mlib\_VectorAdd\_S8C\_Mod, mlib\_VectorAdd\_S8C\_Sat, mlib\_VectorAdd\_S16\_Mod, mlib\_VectorAdd\_S16\_Sat, mlib\_VectorAdd\_S16C\_Mod, mlib\_VectorAdd\_S16C\_Sat, mlib\_VectorAdd\_S32\_Mod, mlib\_VectorAdd\_S32\_Sat, mlib\_VectorAdd\_S32C\_Mod, mlib\_VectorAdd\_S32C\_Sat – vector addition, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VectorAdd_U8_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8C_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8C_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8C_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8C_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_Mod(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_Sat(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32C_Mod(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);
```



```
mllib_status mllib_VectorAdd_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions performs the in-place addition of one vector to another vector.

It uses the following equation:

$$xz[i] = xz[i] + y[i]$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- xz*     Pointer to the first element of the first source and destination vector.
- y*     Pointer to the first element of the second source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorAdd\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorAdd\_U8\_U8\_Mod, mlib\_VectorAdd\_U8\_U8\_Sat,  
 mlib\_VectorAdd\_U8C\_U8C\_Mod, mlib\_VectorAdd\_U8C\_U8C\_Sat,  
 mlib\_VectorAdd\_S8\_S8\_Mod, mlib\_VectorAdd\_S8\_S8\_Sat,  
 mlib\_VectorAdd\_S8C\_S8C\_Mod, mlib\_VectorAdd\_S8C\_S8C\_Sat,  
 mlib\_VectorAdd\_S16\_U8\_Mod, mlib\_VectorAdd\_S16\_U8\_Sat,  
 mlib\_VectorAdd\_S16\_S8\_Mod, mlib\_VectorAdd\_S16\_S8\_Sat,  
 mlib\_VectorAdd\_S16\_S16\_Mod, mlib\_VectorAdd\_S16\_S16\_Sat,  
 mlib\_VectorAdd\_S16C\_U8C\_Mod, mlib\_VectorAdd\_S16C\_U8C\_Sat,  
 mlib\_VectorAdd\_S16C\_S8C\_Mod, mlib\_VectorAdd\_S16C\_S8C\_Sat,  
 mlib\_VectorAdd\_S16C\_S16C\_Mod, mlib\_VectorAdd\_S16C\_S16C\_Sat,  
 mlib\_VectorAdd\_S32\_S16\_Mod, mlib\_VectorAdd\_S32\_S16\_Sat,  
 mlib\_VectorAdd\_S32\_S32\_Mod, mlib\_VectorAdd\_S32\_S32\_Sat,  
 mlib\_VectorAdd\_S32C\_S16C\_Mod, mlib\_VectorAdd\_S32C\_S16C\_Sat,  
 mlib\_VectorAdd\_S32C\_S32C\_Mod, mlib\_VectorAdd\_S32C\_S32C\_Sat – vector addition

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VectorAdd_U8_U8_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8_U8_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8C_U8C_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_U8C_U8C_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8_S8_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8_S8_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8C_S8C_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S8C_S8C_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_U8_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_U8_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_S8_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
```

```

mlib_status mlib_VectorAdd_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_U8C_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_U8C_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_S8C_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_S8C_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_S16_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_S16_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_S32_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32_S32_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32C_S16C_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32C_S16C_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32C_S32C_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAdd_S32C_S32C_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

```

**Description** Each of these functions performs the addition of one vector to another vector.

It uses the following equation:

$$z[i] = x[i] + y[i]$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the first element of the destination vector.
- x*     Pointer to the first element of the first source vector.
- y*     Pointer to the first element of the second source vector.
- n*     Number of elements in the vectors.

**Return Values**   Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**   See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**   [mllib\\_VectorAdd\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorAng\_U8C, mllib\_VectorAng\_S8C, mllib\_VectorAng\_S16C, mllib\_VectorAng\_S32C – vector complex phase (angle)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorAng_U8C(mllib_d64 *a,
 const mllib_u8 *x, mllib_s32 n);

mllib_status mllib_VectorAng_S8C(mllib_d64 *a,
 const mllib_s8 *x, mllib_s32 n);

mllib_status mllib_VectorAng_S16C(mllib_d64 *a,
 const mllib_s16 *x, mllib_s32 n);

mllib_status mllib_VectorAng_S32C(mllib_d64 *a,
 const mllib_s32 *x, mllib_s32 n);
```

**Description** Each of these functions computes the phase vector of a complex vector.

The following equation is used:

$$a[i] = \text{atan}(x[2*i + 1] / x[2*i])$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- a*     Pointer to the destination phase vector.
- x*     Pointer to the source vector
- n*     Number of elements in the vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorAve\_U8, mllib\_VectorAve\_U8C, mllib\_VectorAve\_S8, mllib\_VectorAve\_S8C, mllib\_VectorAve\_S16, mllib\_VectorAve\_S16C, mllib\_VectorAve\_S32, mllib\_VectorAve\_S32C  
– vector average, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VectorAve_U8(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_U8C(mllib_u8 *xz,
 const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S8(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S8C(mllib_s8 *xz,
 const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16C(mllib_s16 *xz,
 const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32C(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions performs an in-place averaging of two vectors.

It uses the following equation:

$$xz[i] = (xz[i] + y[i] + 1) / 2$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

*xz*     Pointer to the first element of the first source and destination vector.

*y*     Pointer to the first element of the second source vector.

*n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

---

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorAve\\_U8\\_U8\(3MLIB\), attributes\(5\)](#)

**Name** mllib\_VectorAve\_U8\_U8, mllib\_VectorAve\_U8C\_U8C, mllib\_VectorAve\_S8\_S8, mllib\_VectorAve\_S8C\_S8C, mllib\_VectorAve\_S16\_U8, mllib\_VectorAve\_S16\_S8, mllib\_VectorAve\_S16\_S16, mllib\_VectorAve\_S16C\_U8C, mllib\_VectorAve\_S16C\_S8C, mllib\_VectorAve\_S16C\_S16C, mllib\_VectorAve\_S32\_S16, mllib\_VectorAve\_S32\_S32, mllib\_VectorAve\_S32C\_S16C, mllib\_VectorAve\_S32C\_S32C – vector average

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorAve_U8_U8(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_U8C_U8C(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S8_S8(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S8C_S8C(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16_U8(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16_S8(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16_S16(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16C_U8C(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16C_S8C(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S16C_S16C(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32_S16(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32_S32(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32C_S16C(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorAve_S32C_S32C(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions computes the average of two vectors.

It uses the following equation:



$$z[i] = (x[i] + y[i] + 1) / 2$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the first element of the destination vector.
- $x$  Pointer to the first element of the first source vector.
- $y$  Pointer to the first element of the second source vector.
- $n$  Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorAve\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorConjRev\_S8C\_S8C\_Sat, mllib\_VectorConjRev\_S16C\_S16C\_Sat, mllib\_VectorConjRev\_S32C\_S32C\_Sat – vector conjugation reversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorConjRev_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, mllib_s32 n);

mllib_status mllib_VectorConjRev_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, mllib_s32 n);

mllib_status mllib_VectorConjRev_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, mllib_s32 n);
```

**Description** Each of these functions computes the complex reversion of a complex vector.

The source and destination vectors must be in the same data type.

The following equation is used:

$$\begin{aligned} z[2*i] &= x[2*(n - 1 - i)] \\ z[2*i + 1] &= -x[2*(n - 1 - i) + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the first element of the destination vector.
- x*     Pointer to the first element of the source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorConj\_S8C\_S8C\_Sat, mllib\_VectorConj\_S16C\_S16C\_Sat, mllib\_VectorConj\_S32C\_S32C\_Sat – vector conjugation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VectorConj_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, mllib_s32 n);

mllib_status mllib_VectorConj_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, mllib_s32 n);

mllib_status mllib_VectorConj_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, mllib_s32 n);
```

**Description** Each of these functions computes the complex conjugate of a complex vector.

The source and destination vectors must be in the same data type.

The following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] \\ z[2*i + 1] &= -x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the first element of the destination vector.
- $x$  Pointer to the first element of the source vector.
- $n$  Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorConj\_S8C\_Sat, mllib\_VectorConj\_S16C\_Sat, mllib\_VectorConj\_S32C\_Sat – vector conjugation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorConj_S8C_Sat(mllib_s8 *xz, mllib_s32 n);
mllib_status mllib_VectorConj_S16C_Sat(mllib_s16 *xz, mllib_s32 n);
mllib_status mllib_VectorConj_S32C_Sat(mllib_s32 *xz, mllib_s32 n);
```

**Description** Each of these functions computes the in-place complex conjugate of a complex vector.

The following equation is used:

$$xz[2*i + 1] = -xz[2*i + 1]$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

*xz*     Pointer to the first element of the source and destination vector.

*n*       Number of elements in the vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorConjSymExt\_S8C\_S8C\_Sat, mllib\_VectorConjSymExt\_S16C\_S16C\_Sat, mllib\_VectorConjSymExt\_S32C\_S32C\_Sat – vector conjugate-symmetric extension

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VectorConjSymExt_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, mllib_s32 n);

mllib_status mllib_VectorConjSymExt_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, mllib_s32 n);

mllib_status mllib_VectorConjSymExt_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, mllib_s32 n);
```

**Description** Each of these functions computes the complex conjugate extension of a complex vector.

The source and destination vectors must be in the same data type.

When  $n$  is even, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] \\ z[2*i + 1] &= -x[2*i + 1] \end{aligned}$$

for  $i = 0, 1, \dots, (n - 1)$ .

$$\begin{aligned} z[2*i] &= x[2*(2*n - 1 - i)] \\ z[2*i + 1] &= -x[2*(2*n - 1 - i) + 1] \end{aligned}$$

for  $i = n, (n + 1), \dots, (2*n - 1)$ .

When  $n$  is odd, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] \\ z[2*i + 1] &= -x[2*i + 1] \end{aligned}$$

for  $i = 0, 1, \dots, (n - 1)$ .

$$\begin{aligned} z[2*i] &= x[2*(2*n - 2 - i)] \\ z[2*i + 1] &= -x[2*(2*n - 2 - i) + 1] \end{aligned}$$

for  $i = n, (n + 1), \dots, (2*n - 2)$ .

**Parameters** Each of the functions takes the following arguments:

$z$     Pointer to the first element of the destination vector.

$x$     Pointer to the first element of the source vector.

$n$     Number of elements in the source vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

## REFERENCE

### Multimedia Library Functions - Part 6

**Name** mllib\_VectorConvert\_U8\_S8\_Mod, mllib\_VectorConvert\_U8\_S16\_Mod, mllib\_VectorConvert\_U8\_S32\_Mod, mllib\_VectorConvert\_S8\_U8\_Mod, mllib\_VectorConvert\_S8\_S16\_Mod, mllib\_VectorConvert\_S8\_S32\_Mod, mllib\_VectorConvert\_S16\_U8\_Mod, mllib\_VectorConvert\_S16\_S8\_Mod, mllib\_VectorConvert\_S16\_S32\_Mod, mllib\_VectorConvert\_S32\_U8\_Mod, mllib\_VectorConvert\_S32\_S8\_Mod, mllib\_VectorConvert\_S32\_S16\_Mod, mllib\_VectorConvert\_U8C\_S8C\_Mod, mllib\_VectorConvert\_U8C\_S16C\_Mod, mllib\_VectorConvert\_U8C\_S32C\_Mod, mllib\_VectorConvert\_S8C\_U8C\_Mod, mllib\_VectorConvert\_S8C\_S16C\_Mod, mllib\_VectorConvert\_S8C\_S32C\_Mod, mllib\_VectorConvert\_S16C\_U8C\_Mod, mllib\_VectorConvert\_S16C\_S8C\_Mod, mllib\_VectorConvert\_S16C\_S32C\_Mod, mllib\_VectorConvert\_S32C\_U8C\_Mod, mllib\_VectorConvert\_S32C\_S8C\_Mod, mllib\_VectorConvert\_S32C\_S16C\_Mod, mllib\_VectorConvert\_U8\_S8\_Sat, mllib\_VectorConvert\_U8\_S16\_Sat, mllib\_VectorConvert\_U8\_S32\_Sat, mllib\_VectorConvert\_S8\_U8\_Sat, mllib\_VectorConvert\_S8\_S16\_Sat, mllib\_VectorConvert\_S8\_S32\_Sat, mllib\_VectorConvert\_S16\_U8\_Sat, mllib\_VectorConvert\_S16\_S8\_Sat, mllib\_VectorConvert\_S16\_S32\_Sat, mllib\_VectorConvert\_S32\_U8\_Sat, mllib\_VectorConvert\_S32\_S8\_Sat, mllib\_VectorConvert\_S32\_S16\_Sat, mllib\_VectorConvert\_U8C\_S8C\_Sat, mllib\_VectorConvert\_U8C\_S16C\_Sat, mllib\_VectorConvert\_U8C\_S32C\_Sat, mllib\_VectorConvert\_S8C\_U8C\_Sat, mllib\_VectorConvert\_S8C\_S16C\_Sat, mllib\_VectorConvert\_S8C\_S32C\_Sat, mllib\_VectorConvert\_S16C\_U8C\_Sat, mllib\_VectorConvert\_S16C\_S8C\_Sat, mllib\_VectorConvert\_S16C\_S32C\_Sat, mllib\_VectorConvert\_S32C\_U8C\_Sat, mllib\_VectorConvert\_S32C\_S8C\_Sat, mllib\_VectorConvert\_S32C\_S16C\_Sat – vector data type convert

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorConvert_U8_S8_Mod(mllib_u8 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_U8_S16_Mod(mllib_u8 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_U8_S32_Mod(mllib_u8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_U8_Mod(mllib_s8 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_S16_Mod(mllib_s8 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_S32_Mod(mllib_s8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 mllib_s32 n);
```



```
mlib_status mlib_VectorConvert_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S16_S32_Mod(mlib_s16 *z, const mlib_s32 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32_U8_Mod(mlib_s32 *z, const mlib_u8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32_S8_Mod(mlib_s32 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_U8C_S8C_Mod(mlib_u8 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_U8C_S16C_Mod(mlib_u8 *z, const mlib_s16 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_U8C_S32C_Mod(mlib_u8 *z, const mlib_s32 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S8C_U8C_Mod(mlib_s8 *z, const mlib_u8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S8C_S16C_Mod(mlib_s8 *z, const mlib_s16 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S8C_S32C_Mod(mlib_s8 *z, const mlib_s32 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S16C_S32C_Mod(mlib_s16 *z, const mlib_s32 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_U8C_Mod(mlib_s32 *z, const mlib_u8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S8C_Mod(mlib_s32 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_U8_S8_Sat(mlib_u8 *z, const mlib_s8 *x,
mlib_s32 n);

mlib_status mlib_VectorConvert_U8_S16_Sat(mlib_u8 *z, const mlib_s16 *x,
mlib_s32 n);
```

```
mllib_status mllib_VectorConvert_U8_S32_Sat(mllib_u8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_U8_Sat(mllib_s8 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_S16_Sat(mllib_s8 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8_S32_Sat(mllib_s8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16_S8_Sat(mllib_s16 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16_S32_Sat(mllib_s16 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S32_U8_Sat(mllib_s32 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S32_S8_Sat(mllib_s32 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S32_S16_Sat(mllib_s32 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_U8C_S8C_Sat(mllib_u8 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_U8C_S16C_Sat(mllib_u8 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_U8C_S32C_Sat(mllib_u8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8C_U8C_Sat(mllib_s8 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8C_S16C_Sat(mllib_s8 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S8C_S32C_Sat(mllib_s8 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorConvert_S16C_S32C_Sat(mllib_s16 *z, const mllib_s32 *x,
 mllib_s32 n);
```

```

mlib_status mlib_VectorConvert_S32C_U8C_Sat(mlib_s32 *z, const mlib_u8 *x,
 mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S8C_Sat(mlib_s32 *z, const mlib_s8 *x,
 mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
 mlib_s32 n);

```

**Description** Each of these functions copies data from one vector to another vector, of different data types.

For real data, the following equation is used:

$$z[i] = x[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] \\ z[2*i + 1] &= x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

See the following tables for available variations of the data type convert function.

| Type[*] | U8 | S8 | S16 | S32 |
|---------|----|----|-----|-----|
| U8      |    | Y  | Y   | Y   |
| S8      | Y  |    | Y   | Y   |
| S16     | Y  | Y  |     | Y   |
| S32     | Y  | Y  | Y   |     |

| Type[*] | U8C | S8C | S16C | S32C |
|---------|-----|-----|------|------|
| U8C     |     | Y   | Y    | Y    |
| S8C     | Y   |     | Y    | Y    |
| S16C    | Y   | Y   |      | Y    |
| S32C    | Y   | Y   | Y    |      |

[\*] Each row represents a source data type. Each column represents a destination data type.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the first element of the destination vector.
- x*     Pointer to the first element of the source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorCopy\_U8, mllib\_VectorCopy\_U8C, mllib\_VectorCopy\_S8, mllib\_VectorCopy\_S8C, mllib\_VectorCopy\_S16, mllib\_VectorCopy\_S16C, mllib\_VectorCopy\_S32, mllib\_VectorCopy\_S32C – vector copy

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorCopy_U8(mllib_u8 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_U8C(mllib_u8 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S8(mllib_s8 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S8C(mllib_s8 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S16(mllib_s16 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S16C(mllib_s16 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S32(mllib_s32 *z, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorCopy_S32C(mllib_s32 *z, const mllib_s32 *x,
 mllib_s32 n);
```

**Description** Each of these functions copies one vector to another vector of the same data type.

The input and output vectors must be in the same data type.

For real data, the following equation is used:

$$z[i] = x[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] \\ z[2*i + 1] &= x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- $z$      Pointer to the first element of the destination vector.
- $x$      Pointer to the first element of the source vector.
- $n$      Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorDistance\_U8\_Sat, mllib\_VectorDistance\_S8\_Sat, mllib\_VectorDistance\_S16\_Sat, mllib\_VectorDistance\_S32\_Sat – vector Euclidean distance

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorDistance_U8_Sat(mllib_d64 *z, const mllib_u8 *x,
 const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorDistance_S8_Sat(mllib_d64 *z, const mllib_s8 *x,
 const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorDistance_S16_Sat(mllib_d64 *z, const mllib_s16 *x,
 const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorDistance_S32_Sat(mllib_d64 *z, const mllib_s32 *x,
 const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions computes the Euclidean distances between two vectors.

The following equation is used:

$$z[0] = \{ \sum_{i=0}^{n-1} (x[i] - y[i])**2 \}**0.5$$

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the distance between the two vectors.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorDotProd\_U8\_Sat, mllib\_VectorDotProd\_U8C\_Sat, mllib\_VectorDotProd\_S8\_Sat, mllib\_VectorDotProd\_S8C\_Sat, mllib\_VectorDotProd\_S16\_Sat, mllib\_VectorDotProd\_S16C\_Sat, mllib\_VectorDotProd\_S32\_Sat, mllib\_VectorDotProd\_S32C\_Sat – vector dot product (inner product)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorDotProd_U8_Sat(mllib_d64 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_U8C_Sat(mllib_d64 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S8_Sat(mllib_d64 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S8C_Sat(mllib_d64 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S16_Sat(mllib_d64 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S16C_Sat(mllib_d64 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S32_Sat(mllib_d64 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorDotProd_S32C_Sat(mllib_d64 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions computes the dot product of two vectors, defined by the following equation:

$$Z = X \cdot Y^*$$

where  $Y^*$  is the conjugate of the  $Y$  vector.

For real data, the following equation is used:

$$z[0] = \sum_{i=0}^{n-1} (x[i] * y[i])$$

For complex data, the following equation is used:

$$z[0] = \sum_{i=0}^{n-1} (x[2*i] * y[2*i] + x[2*i + 1] * y[2*i + 1])$$



$$z[1] = \sum_{i=0} (x[2*i + 1]*y[2*i] - x[2*i]*y[2*i + 1])$$

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the dot product of the two vectors.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorMag\_U8C, mllib\_VectorMag\_S8C, mllib\_VectorMag\_S16C, mllib\_VectorMag\_S32C – vector complex magnitude

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VectorMag_U8C(mllib_d64 *m, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorMag_S8C(mllib_d64 *m, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorMag_S16C(mllib_d64 *m, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorMag_S32C(mllib_d64 *m, const mllib_s32 *x,
 mllib_s32 n);
```

**Description** Each of these functions computes the magnitude vector of the complex input vector.

The following equation is used:

$$m[i] = (x[2*i]**2 + x[2*i + 1]**2)**0.5$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- m*     Pointer to the destination magnitude vector.
- x*     Pointer to the source vector
- n*     Number of elements in the vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorMaximumMag\_U8C, mllib\_VectorMaximumMag\_S8C, mllib\_VectorMaximumMag\_S16C, mllib\_VectorMaximumMag\_S32C, mllib\_VectorMaximumMag\_F32C, mllib\_VectorMaximumMag\_D64C – find the first element with the maximum magnitude in a vector

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_VectorMaximumMag_U8C(mllib_u8 *max,  
const mllib_u8 *x, mllib_s32 n);  
  
mllib_status mllib_VectorMaximumMag_S8C(mllib_s8 *max,  
const mllib_s8 *x, mllib_s32 n);  
  
mllib_status mllib_VectorMaximumMag_S16C(mllib_s16 *max,  
const mllib_s16 *x, mllib_s32 n);  
  
mllib_status mllib_VectorMaximumMag_S32C(mllib_s32 *max,  
const mllib_s32 *x, mllib_s32 n);  
  
mllib_status mllib_VectorMaximumMag_F32C(mllib_f32 *max,  
const mllib_f32 *x, mllib_s32 n);  
  
mllib_status mllib_VectorMaximumMag_D64C(mllib_d64 *max,  
const mllib_d64 *x, mllib_s32 n);`

**Description** Each of these functions finds the first element with the maximum magnitude in a complex vector, then puts the real and imaginary parts of it into max[0] and max[1], respectively.

**Parameters** Each of the functions takes the following arguments:

*max*     Pointer to the first element with the maximum magnitude.

*x*        Pointer to the first element of the source vector.

*n*        Number of elements in the source vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMinimumMag\\_U8C\(3MLIB\)](#), [mllib\\_MatrixMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_MatrixMinimumMag\\_U8C\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMaximum\_U8, mllib\_VectorMaximum\_S8, mllib\_VectorMaximum\_S16, mllib\_VectorMaximum\_S32, mllib\_VectorMaximum\_F32, mllib\_VectorMaximum\_D64 – find the maximum value in a vector

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_VectorMaximum_U8(mllib_u8 *max, const mllib_u8 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMaximum_S8(mllib_s8 *max, const mllib_s8 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMaximum_S16(mllib_s16 *max, const mllib_s16 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMaximum_S32(mllib_s32 *max, const mllib_s32 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMaximum_F32(mllib_f32 *max, const mllib_f32 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMaximum_D64(mllib_d64 *max, const mllib_d64 *x,  
 mllib_s32 n);`

**Description** Each of these functions finds the maximum value of all elements in a vector.  
  
The following equation is used:

$$\text{max}[0] = \text{MAX}\{ x[i] \mid i = 0, 1, \dots, (n - 1) \}$$

**Parameters** Each of the functions takes the following arguments:

*max*      Pointer to the maximum value.

*x*        Pointer to the first element of the source vector.

*n*        Number of elements in the source vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMinimum\\_U8\(3MLIB\)](#), [mllib\\_MatrixMaximum\\_U8\(3MLIB\)](#),  
[mllib\\_MatrixMinimum\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMerge\_U8C\_U8, mllib\_VectorMerge\_S8C\_S8, mllib\_VectorMerge\_S16C\_S16, mllib\_VectorMerge\_S32C\_S32 – vector merge

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_VectorMerge_U8C_U8(mllib_u8 *z, const mllib_u8 *r,
 const mllib_u8 *i, mllib_s32 n);

mllib_status mllib_VectorMerge_S8C_S8(mllib_s8 *z, const mllib_s8 *r,
 const mllib_s8 *i, mllib_s32 n);

mllib_status mllib_VectorMerge_S16C_S16(mllib_s16 *z, const mllib_s16 *r,
 const mllib_s16 *i, mllib_s32 n);

mllib_status mllib_VectorMerge_S32C_S32(mllib_s32 *z, const mllib_s32 *r,
 const mllib_s32 *i, mllib_s32 n);
```

**Description** Each of these functions computes the complex vector from two vectors representing the real and imaginary parts.

The following equation is used:

$$\begin{aligned} z[2*k] &= r[k] \\ z[2*k + 1] &= i[k] \end{aligned}$$

where  $k = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

*z* Pointer to the first complex element of the destination vector.  $z[2*k]$  contains the real part, and  $z[2*k + 1]$  contains the imaginary part.

*r* Pointer to the first element of the real part.

*i* Pointer to the first element of the imaginary part.

*n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorSplit\\_U8\\_U8C\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMinimumMag\_U8C, mllib\_VectorMinimumMag\_S8C, mllib\_VectorMinimumMag\_S16C, mllib\_VectorMinimumMag\_S32C, mllib\_VectorMinimumMag\_F32C, mllib\_VectorMinimumMag\_D64C – find the first element with the minimum magnitude in a vector

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VectorMinimumMag_U8C(mllib_u8 *min, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorMinimumMag_S8C(mllib_s8 *min, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorMinimumMag_S16C(mllib_s16 *min, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorMinimumMag_S32C(mllib_s32 *min, const mllib_s32 *x,
 mllib_s32 n);

mllib_status mllib_VectorMinimumMag_F32C(mllib_f32 *min, const mllib_f32 *x,
 mllib_s32 n);

mllib_status mllib_VectorMinimumMag_D64C(mllib_d64 *min, const mllib_d64 *x,
 mllib_s32 n);
```

**Description** Each of these functions finds the first element with the minimum magnitude in a complex vector, then puts the real and imaginary parts of it into `min[0]` and `min[1]`, respectively.

**Parameters** Each of the functions takes the following arguments:

*min*     Pointer to the first element with the minimum magnitude.  
*x*        Pointer to the first element of the source vector.  
*n*        Number of elements in the source vector.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_MatrixMaximumMag\\_U8C\(3MLIB\)](#), [mllib\\_MatrixMinimumMag\\_U8C\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMinimum\_U8, mllib\_VectorMinimum\_S8, mllib\_VectorMinimum\_S16, mllib\_VectorMinimum\_S32, mllib\_VectorMinimum\_F32, mllib\_VectorMinimum\_D64 – find the minimum value in a vector

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_VectorMinimum_U8(mllib_u8 *min, const mllib_u8 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMinimum_S8(mllib_s8 *min, const mllib_s8 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMinimum_S16(mllib_s16 *min, const mllib_s16 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMinimum_S32(mllib_s32 *min, const mllib_s32 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMinimum_F32(mllib_f32 *min, const mllib_f32 *x,  
 mllib_s32 n);  
  
mllib_status mllib_VectorMinimum_D64(mllib_d64 *min, const mllib_d64 *x,  
 mllib_s32 n);`

**Description** Each of these functions finds the minimum value of all elements in a vector.  
  
The following equation is used:

$$\max[0] = \text{MIN}\{ x[i] \mid i = 0, 1, \dots, (n - 1) \}$$

**Parameters** Each of the functions takes the following arguments:

*min*     Pointer to the minimum value.

*x*       Pointer to the first element of the source vector.

*n*       Number of elements in the source vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMaximum\\_U8\(3MLIB\)](#), [mllib\\_MatrixMaximum\\_U8\(3MLIB\)](#), [mllib\\_MatrixMinimum\\_U8\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_VectorMulMShift\_S16\_S16\_Mod, mlib\_VectorMulMShift\_S16\_S16\_Sat, mlib\_VectorMulMShift\_S16C\_S16C\_Mod, mlib\_VectorMulMShift\_S16C\_S16C\_Sat – multiplication of vector by matrix with shifting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VectorMulMShift_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
 mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulMShift_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
 mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulMShift_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
 mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulMShift_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
 mlib_s32 n, mlib_s32 shift);
```

**Description** Each of these functions multiplies a vector by a matrix and shifts the results.

For real data, the following equation is used:

$$z[i] = \left\{ \sum_{j=0}^{m-1} (x[j] * y[j*m + i]) \right\} * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*i] = \left\{ \sum_{j=0}^{m-1} (xR*yR - xI*yI) \right\} * 2^{**(-shift)}$$

$$z[2*i + 1] = \left\{ \sum_{j=0}^{m-1} (xR*yI + xI*yR) \right\} * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ , and

```
xR = x[2*j]
xI = x[2*j + 1]
yR = y[2*(j*m + i)]
yI = y[2*(j*m + i) + 1]
```

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the source vector.
- y* Pointer to the first element of the source matrix.
- m* Number of rows in the matrix, and number of elements in the source vector.
- n* Number of columns in the matrix, and number of elements in the destination vector.
- shift* Right shifting factor.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulM\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulM\_U8\_U8\_Mod, mllib\_VectorMulM\_U8\_U8\_Sat, mllib\_VectorMulM\_U8C\_U8C\_Mod, mllib\_VectorMulM\_U8C\_U8C\_Sat, mllib\_VectorMulM\_S8\_S8\_Mod, mllib\_VectorMulM\_S8\_S8\_Sat, mllib\_VectorMulM\_S8C\_S8C\_Mod, mllib\_VectorMulM\_S8C\_S8C\_Sat, mllib\_VectorMulM\_S16\_U8\_Mod, mllib\_VectorMulM\_S16\_U8\_Sat, mllib\_VectorMulM\_S16\_S8\_Mod, mllib\_VectorMulM\_S16\_S8\_Sat, mllib\_VectorMulM\_S16\_S16\_Mod, mllib\_VectorMulM\_S16\_S16\_Sat, mllib\_VectorMulM\_S16C\_U8C\_Mod, mllib\_VectorMulM\_S16C\_U8C\_Sat, mllib\_VectorMulM\_S16C\_S8C\_Mod, mllib\_VectorMulM\_S16C\_S8C\_Sat, mllib\_VectorMulM\_S16C\_S16C\_Mod, mllib\_VectorMulM\_S16C\_S16C\_Sat, mllib\_VectorMulM\_S32\_S16\_Mod, mllib\_VectorMulM\_S32\_S16\_Sat, mllib\_VectorMulM\_S32\_S32\_Mod, mllib\_VectorMulM\_S32\_S32\_Sat, mllib\_VectorMulM\_S32C\_S16C\_Mod, mllib\_VectorMulM\_S32C\_S16C\_Sat, mllib\_VectorMulM\_S32C\_S32C\_Mod, mllib\_VectorMulM\_S32C\_S32C\_Sat – multiplication of vector by matrix

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorMulM_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);
```

```
mllib_status mllib_VectorMulM_S16_S8_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16_S16_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16_S16_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_U8C_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_S8C_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_S16C_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S16C_S16C_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32_S16_Mod(mlib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32_S16_Sat(mlib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32_S32_Mod(mlib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32_S32_Sat(mlib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32C_S16C_Mod(mlib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32C_S16C_Sat(mlib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32C_S32C_Mod(mlib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);

mllib_status mllib_VectorMulM_S32C_S32C_Sat(mlib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *y, mlib_s32 m, mlib_s32 n);
```

**Description** Each of these functions multiplies a vector by a matrix.

For real data, the following equation is used:

$$z[i] = \sum_{j=0}^{m-1} (x[j] * y[j*m + i])$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*i] = \sum_{j=0}^{m-1} (xR*yR - xI*yI)$$

$$z[2*i + 1] = \sum_{j=0}^{m-1} (xR*yI + xI*yR)$$

where  $i = 0, 1, \dots, (n - 1)$ , and

$$\begin{aligned} xR &= x[2*j] \\ xI &= x[2*j + 1] \\ yR &= y[2*(j*m + i)] \\ yI &= y[2*(j*m + i) + 1] \end{aligned}$$

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the source vector.
- y* Pointer to the first element of the source matrix.
- m* Number of rows in the matrix, and number of elements in the source vector.
- n* Number of columns in the matrix, and number of elements in the destination vector.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulMShift\\_S16\\_S16\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulSAdd\_U8\_Mod, mllib\_VectorMulSAdd\_U8\_Sat, mllib\_VectorMulSAdd\_U8C\_Mod, mllib\_VectorMulSAdd\_U8C\_Sat, mllib\_VectorMulSAdd\_S8\_Mod, mllib\_VectorMulSAdd\_S8\_Sat, mllib\_VectorMulSAdd\_S8C\_Mod, mllib\_VectorMulSAdd\_S8C\_Sat, mllib\_VectorMulSAdd\_S16\_Mod, mllib\_VectorMulSAdd\_S16\_Sat, mllib\_VectorMulSAdd\_S16C\_Mod, mllib\_VectorMulSAdd\_S16C\_Sat, mllib\_VectorMulSAdd\_S32\_Mod, mllib\_VectorMulSAdd\_S32\_Sat, mllib\_VectorMulSAdd\_S32C\_Mod, mllib\_VectorMulSAdd\_S32C\_Sat – vector multiplication by scalar plus addition, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mllib_VectorMulSAdd_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *y, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *y, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *y, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *y, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *y, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *y, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *y, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *y, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *y, const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *y, const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *y, const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *y, const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *y, const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *y, const mllib_s32 *c, mllib_s32 n);

```

```

mllib_status mllib_VectorMulSAdd_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *y, const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, const mllib_s32 *c, mllib_s32 n);

```

**Description** Each of these functions computes an in-place multiplication of a vector by a scalar and adds the result to another vector.

For real data, the following equation is used:

$$xz[i] = xz[i] + y[i]*c[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned}
 xz[2*i] &= xz[2*i] + y[2*i]*c[0] - y[2*i + 1]*c[1] \\
 xz[2*i + 1] &= xz[2*i + 1] + y[2*i]*c[1] + y[2*i + 1]*c[0]
 \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the first source and destination vector.
- y* Pointer to the first element of the second source vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulSAdd\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorMulSAdd\_U8\_U8\_Mod, mlib\_VectorMulSAdd\_U8\_U8\_Sat, mlib\_VectorMulSAdd\_U8C\_U8C\_Mod, mlib\_VectorMulSAdd\_U8C\_U8C\_Sat, mlib\_VectorMulSAdd\_S8\_S8\_Mod, mlib\_VectorMulSAdd\_S8\_S8\_Sat, mlib\_VectorMulSAdd\_S8C\_S8C\_Mod, mlib\_VectorMulSAdd\_S8C\_S8C\_Sat, mlib\_VectorMulSAdd\_S16\_U8\_Mod, mlib\_VectorMulSAdd\_S16\_U8\_Sat, mlib\_VectorMulSAdd\_S16\_S8\_Mod, mlib\_VectorMulSAdd\_S16\_S8\_Sat, mlib\_VectorMulSAdd\_S16\_S16\_Mod, mlib\_VectorMulSAdd\_S16\_S16\_Sat, mlib\_VectorMulSAdd\_S16C\_U8C\_Mod, mlib\_VectorMulSAdd\_S16C\_U8C\_Sat, mlib\_VectorMulSAdd\_S16C\_S8C\_Mod, mlib\_VectorMulSAdd\_S16C\_S8C\_Sat, mlib\_VectorMulSAdd\_S16C\_S16C\_Mod, mlib\_VectorMulSAdd\_S16C\_S16C\_Sat, mlib\_VectorMulSAdd\_S32\_S16\_Mod, mlib\_VectorMulSAdd\_S32\_S16\_Sat, mlib\_VectorMulSAdd\_S32\_S32\_Mod, mlib\_VectorMulSAdd\_S32\_S32\_Sat, mlib\_VectorMulSAdd\_S32C\_S16C\_Mod, mlib\_VectorMulSAdd\_S32C\_S16C\_Sat, mlib\_VectorMulSAdd\_S32C\_S32C\_Mod, mlib\_VectorMulSAdd\_S32C\_S32C\_Sat – vector multiplication by scalar plus addition

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mlib.h>

```
mlib_status mlib_VectorMulSAdd_U8_U8_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_U8_U8_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_U8C_U8C_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_U8C_U8C_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_S8_S8_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_S8_S8_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_S8C_S8C_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);
```

```
mlib_status mlib_VectorMulSAdd_S8C_S8C_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);
```



```

mllib_status mllib_VectorMulSAdd_S16_U8_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_U8_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_S8_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_U8C_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_U8C_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_S8C_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_S8C_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S32_S16_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

mllib_status mllib_VectorMulSAdd_S32_S16_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
 mlib_s32 n);

```

```
mllib_status mllib_VectorMulSAdd_S32_S32_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, const mllib_s32 *c,
 mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32_S32_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, const mllib_s32 *c,
 mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32C_S16C_Mod(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32C_S16C_Sat(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, const mllib_s32 *c,
 mllib_s32 n);

mllib_status mllib_VectorMulSAdd_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, const mllib_s32 *c,
 mllib_s32 n);
```

**Description** Each of these functions multiplies a vector by a scalar and adds the result to another vector.

For real data, the following equation is used:

$$z[i] = x[i] + y[i]*c[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i] + y[2*i]*c[0] - y[2*i + 1]*c[1] \\ z[2*i + 1] &= x[2*i + 1] + y[2*i]*c[1] + y[2*i + 1]*c[0] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorMulSAdd\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorMulShift\_U8\_Mod, mlib\_VectorMulShift\_U8\_Sat, mlib\_VectorMulShift\_U8C\_Mod, mlib\_VectorMulShift\_U8C\_Sat, mlib\_VectorMulShift\_S8\_Mod, mlib\_VectorMulShift\_S8\_Sat, mlib\_VectorMulShift\_S8C\_Mod, mlib\_VectorMulShift\_S8C\_Sat, mlib\_VectorMulShift\_S16\_Mod, mlib\_VectorMulShift\_S16\_Sat, mlib\_VectorMulShift\_S16C\_Mod, mlib\_VectorMulShift\_S16C\_Sat, mlib\_VectorMulShift\_S32\_Mod, mlib\_VectorMulShift\_S32\_Sat, mlib\_VectorMulShift\_S32C\_Mod, mlib\_VectorMulShift\_S32C\_Sat – vector multiplication with shifting, in place

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VectorMulShift_U8_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_U8_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_U8C_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_U8C_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S8_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S8_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S8C_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S8C_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S16_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S16_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S16C_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S16C_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S32_Mod(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S32_Sat(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
```

```
mllib_status mllib_VectorMulShift_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulShift_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n, mllib_s32 shift);
```

**Description** Each of these functions performs an in-place multiplication of two vectors and shifts the result.

For real data, the following equation is used:

$$xz[i] = xz[i] * y[i] * 2^{*(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} tmp &= xz[2*i] \\ xz[2*i] &= (tmp*y[2*i] - xz[2*i + 1]*y[2*i + 1]) * 2^{*(-shift)} \\ xz[2*i + 1] &= (tmp*y[2*i + 1] + xz[2*i + 1]*y[2*i]) * 2^{*(-shift)} \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the first source and result vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in each vector.
- shift* Right shifting factor. The ranges of valid *shift* are:
  - $1 \leq shift \leq 8$  for U8, S8, U8C, S8C types
  - $1 \leq shift \leq 16$  for S16, S16C types
  - $1 \leq shift \leq 31$  for S32, S32C types

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulShift\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulShift\_U8\_U8\_Mod, mllib\_VectorMulShift\_U8\_U8\_Sat, mllib\_VectorMulShift\_U8C\_U8C\_Mod, mllib\_VectorMulShift\_U8C\_U8C\_Sat, mllib\_VectorMulShift\_S8\_S8\_Mod, mllib\_VectorMulShift\_S8\_S8\_Sat, mllib\_VectorMulShift\_S8C\_S8C\_Mod, mllib\_VectorMulShift\_S8C\_S8C\_Sat, mllib\_VectorMulShift\_S16\_S16\_Mod, mllib\_VectorMulShift\_S16\_S16\_Sat, mllib\_VectorMulShift\_S16C\_S16C\_Mod, mllib\_VectorMulShift\_S16C\_S16C\_Sat, mllib\_VectorMulShift\_S32\_S32\_Mod, mllib\_VectorMulShift\_S32\_S32\_Sat, mllib\_VectorMulShift\_S32C\_S32C\_Mod, mllib\_VectorMulShift\_S32C\_S32C\_Sat – vector multiplication with shifting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mllib_VectorMulShift_U8_U8_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_U8_U8_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_U8C_U8C_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_U8C_U8C_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S8_S8_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S8_S8_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S8C_S8C_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S16_S16_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S16_S16_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S16C_S16C_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S32_S32_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mlib_s32 n, mlib_s32 shift);

mllib_status mllib_VectorMulShift_S32_S32_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mlib_s32 n, mlib_s32 shift);

```

```

mllib_status mllib_VectorMulShift_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulShift_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n, mllib_s32 shift);

```

**Description** Each of these functions performs a multiplication of two vectors, shifts the result, and puts it into a third vector.

For real data, the following equation is used:

$$z[i] = x[i] * y[i] * 2^{**(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned}
 z[2*i] &= (x[2*i]*y[2*i] - x[2*i + 1]*y[2*i + 1]) * 2^{**(-shift)} \\
 z[2*i + 1] &= (x[2*i]*y[2*i + 1] + x[2*i + 1]*y[2*i]) * 2^{**(-shift)}
 \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the result vector.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in each vector.
- shift* Right shifting factor. The ranges of valid shift are:
  - $1 \leq \text{shift} \leq 8$  for U8, S8, U8C, S8C types
  - $1 \leq \text{shift} \leq 16$  for S16, S16C types
  - $1 \leq \text{shift} \leq 31$  for S32, S32C types

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulShift\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulSShift\_U8\_Mod, mllib\_VectorMulSShift\_U8\_Sat, mllib\_VectorMulSShift\_U8C\_Mod, mllib\_VectorMulSShift\_U8C\_Sat, mllib\_VectorMulSShift\_S8\_Mod, mllib\_VectorMulSShift\_S8\_Sat, mllib\_VectorMulSShift\_S8C\_Mod, mllib\_VectorMulSShift\_S8C\_Sat, mllib\_VectorMulSShift\_S16\_Mod, mllib\_VectorMulSShift\_S16\_Sat, mllib\_VectorMulSShift\_S16C\_Mod, mllib\_VectorMulSShift\_S16C\_Sat, mllib\_VectorMulSShift\_S32\_Mod, mllib\_VectorMulSShift\_S32\_Sat, mllib\_VectorMulSShift\_S32C\_Mod, mllib\_VectorMulSShift\_S32C\_Sat – vector multiplication by scalar plus shifting, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorMulSShift_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n, mllib_s32 shift);
```



```

mlib_status mlib_VectorMulSShift_S32C_Mod(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulSShift_S32C_Sat(mlib_s32 *xz,
 const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

```

**Description** Each of these functions performs an in-place multiplication of a vector by a scalar and shifts the result.

For real data, the following equation is used:

$$xz[i] = xz[i] * c[0] * 2^{*(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned}
tmp &= xz[2*i] \\
xz[2*i] &= (tmp*c[0] - xz[2*i + 1]*c[1]) * 2^{*(-shift)} \\
xz[2*i + 1] &= (tmp*c[1] + xz[2*i + 1]*c[0]) * 2^{*(-shift)}
\end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

*xz* Pointer to the first element of the source and result vector.

*c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.

*n* Number of elements in each vector.

*shift* Right shifting factor. The ranges of valid *shift* are:

$$\begin{aligned}
1 \leq \text{shift} \leq 8 & \quad \text{for U8, S8, U8C, S8C types} \\
1 \leq \text{shift} \leq 16 & \quad \text{for S16, S16C types} \\
1 \leq \text{shift} \leq 31 & \quad \text{for S32, S32C types}
\end{aligned}$$

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorMulSShift\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorMulSShift\_U8\_U8\_Mod, mlib\_VectorMulSShift\_U8\_U8\_Sat, mlib\_VectorMulSShift\_U8C\_U8C\_Mod, mlib\_VectorMulSShift\_U8C\_U8C\_Sat, mlib\_VectorMulSShift\_S8\_S8\_Mod, mlib\_VectorMulSShift\_S8\_S8\_Sat, mlib\_VectorMulSShift\_S8C\_S8C\_Mod, mlib\_VectorMulSShift\_S8C\_S8C\_Sat, mlib\_VectorMulSShift\_S16\_S16\_Mod, mlib\_VectorMulSShift\_S16\_S16\_Sat, mlib\_VectorMulSShift\_S16C\_S16C\_Mod, mlib\_VectorMulSShift\_S16C\_S16C\_Sat, mlib\_VectorMulSShift\_S32\_S32\_Mod, mlib\_VectorMulSShift\_S32\_S32\_Sat, mlib\_VectorMulSShift\_S32C\_S32C\_Mod, mlib\_VectorMulSShift\_S32C\_S32C\_Sat – vector multiplication by scalar plus shifting

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```

mllib_status mlib_VectorMulSShift_U8_U8_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_U8_U8_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_U8C_U8C_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_U8C_U8C_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S8_S8_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S8_S8_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S8C_S8C_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S8C_S8C_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S32_S32_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

mllib_status mlib_VectorMulSShift_S32_S32_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

```

```
mllib_status mllib_VectorMulSShift_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 n, mllib_s32 shift);

mllib_status mllib_VectorMulSShift_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *c, mllib_s32 n, mllib_s32 shift);
```

**Description** Each of these functions performs a multiplication of a vector by a scalar and shifts the result.

For real data, the following equation is used:

$$z[i] = x[i] * c[0] * 2^{*(-shift)}$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= (x[2*i]*c[0] - x[2*i + 1]*c[1]) * 2^{*(-shift)} \\ z[2*i + 1] &= (x[2*i]*c[1] + x[2*i + 1]*c[0]) * 2^{*(-shift)} \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the result vector.
- x* Pointer to the first element of the source vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.
- n* Number of elements in each vector.
- shift* Right shifting factor. The ranges of valid shift are:
  - $1 \leq shift \leq 8$  for U8, S8, U8C, S8C types
  - $1 \leq shift \leq 16$  for S16, S16C types
  - $1 \leq shift \leq 31$  for S32, S32C types

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulSShift\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulS\_U8\_Mod, mllib\_VectorMulS\_U8\_Sat, mllib\_VectorMulS\_U8C\_Mod, mllib\_VectorMulS\_U8C\_Sat, mllib\_VectorMulS\_S8\_Mod, mllib\_VectorMulS\_S8\_Sat, mllib\_VectorMulS\_S8C\_Mod, mllib\_VectorMulS\_S8C\_Sat, mllib\_VectorMulS\_S16\_Mod, mllib\_VectorMulS\_S16\_Sat, mllib\_VectorMulS\_S16C\_Mod, mllib\_VectorMulS\_S16C\_Sat, mllib\_VectorMulS\_S32\_Mod, mllib\_VectorMulS\_S32\_Sat, mllib\_VectorMulS\_S32C\_Mod, mllib\_VectorMulS\_S32C\_Sat – vector multiplication by scalar, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorMulS_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorMulS_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);
```

```
mllib_status mllib_VectorMulS_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);
```

**Description** Each of these functions computes an in-place multiplication of a vector by a scalar.

For real data, the following equation is used:

$$xz[i] = xz[i] * c[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} tmp &= xz[2*i] \\ xz[2*i] &= tmp*c[0] - xz[2*i + 1]*c[1] \\ xz[2*i + 1] &= tmp*c[1] + xz[2*i + 1]*c[0] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the source and destination vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorMulS\_U8\_U8\_Mod, mllib\_VectorMulS\_U8\_U8\_Sat, mllib\_VectorMulS\_U8C\_U8C\_Mod, mllib\_VectorMulS\_U8C\_U8C\_Sat, mllib\_VectorMulS\_S8\_S8\_Mod, mllib\_VectorMulS\_S8\_S8\_Sat, mllib\_VectorMulS\_S8C\_S8C\_Mod, mllib\_VectorMulS\_S8C\_S8C\_Sat, mllib\_VectorMulS\_S16\_U8\_Mod, mllib\_VectorMulS\_S16\_U8\_Sat, mllib\_VectorMulS\_S16\_S8\_Mod, mllib\_VectorMulS\_S16\_S8\_Sat, mllib\_VectorMulS\_S16\_S16\_Mod, mllib\_VectorMulS\_S16\_S16\_Sat, mllib\_VectorMulS\_S16C\_U8C\_Mod, mllib\_VectorMulS\_S16C\_U8C\_Sat, mllib\_VectorMulS\_S16C\_S8C\_Mod, mllib\_VectorMulS\_S16C\_S8C\_Sat, mllib\_VectorMulS\_S16C\_S16C\_Mod, mllib\_VectorMulS\_S16C\_S16C\_Sat, mllib\_VectorMulS\_S32\_S16\_Mod, mllib\_VectorMulS\_S32\_S16\_Sat, mllib\_VectorMulS\_S32\_S32\_Mod, mllib\_VectorMulS\_S32\_S32\_Sat, mllib\_VectorMulS\_S32C\_S16C\_Mod, mllib\_VectorMulS\_S32C\_S16C\_Sat, mllib\_VectorMulS\_S32C\_S32C\_Mod, mllib\_VectorMulS\_S32C\_S32C\_Sat – vector multiplication by scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorMulS_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorMulS_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *c, mlib_s32 n);
```

```

mlib_status mlib_VectorMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x,
 const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
 const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
 const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
 const mlib_s32 *c, mlib_s32 n);

```

**Description** Each of these functions multiplies a vector by a scalar.

For real data, the following equation is used:

$$z[i] = x[i] * c[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i]*c[0] - x[2*i + 1]*c[1] \\ z[2*i + 1] &= x[2*i]*c[1] + x[2*i + 1]*c[0] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the source vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the real part of the scalar, and *c*[1] contains the imaginary part of the scalar.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMulS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_VectorMul\_U8\_Mod, mlib\_VectorMul\_U8\_Sat, mlib\_VectorMul\_U8C\_Mod, mlib\_VectorMul\_U8C\_Sat, mlib\_VectorMul\_S8\_Mod, mlib\_VectorMul\_S8\_Sat, mlib\_VectorMul\_S8C\_Mod, mlib\_VectorMul\_S8C\_Sat, mlib\_VectorMul\_S16\_Mod, mlib\_VectorMul\_S16\_Sat, mlib\_VectorMul\_S16C\_Mod, mlib\_VectorMul\_S16C\_Sat, mlib\_VectorMul\_S32\_Mod, mlib\_VectorMul\_S32\_Sat, mlib\_VectorMul\_S32C\_Mod, mlib\_VectorMul\_S32C\_Sat – vector multiplication, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VectorMul_U8_Mod(mlib_u8 *xz, const mlib_u8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_U8_Sat(mlib_u8 *xz, const mlib_u8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_U8C_Mod(mlib_u8 *xz, const mlib_u8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_U8C_Sat(mlib_u8 *xz, const mlib_u8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S8_Mod(mlib_s8 *xz, const mlib_s8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S8_Sat(mlib_s8 *xz, const mlib_s8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S8C_Mod(mlib_s8 *xz, const mlib_s8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S8C_Sat(mlib_s8 *xz, const mlib_s8 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S16_Mod(mlib_s16 *xz, const mlib_s16 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S16_Sat(mlib_s16 *xz, const mlib_s16 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S16C_Mod(mlib_s16 *xz, const mlib_s16 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S16C_Sat(mlib_s16 *xz, const mlib_s16 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S32_Mod(mlib_s32 *xz, const mlib_s32 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S32_Sat(mlib_s32 *xz, const mlib_s32 *y,
 mlib_s32 n);

mlib_status mlib_VectorMul_S32C_Mod(mlib_s32 *xz, const mlib_s32 *y,
 mlib_s32 n);
```

```
mllib_status mllib_VectorMul_S32C_Sat(mllib_s32 *xz, const mllib_s32 *y,
 mllib_s32 n);
```

**Description** Each of these functions performs an in-place multiplication of one vector by another vector.

For real data, the following equation is used:

$$xz[i] = xz[i] * y[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

```
tmp = xz[2*i]
xz[2*i] = tmp*y[2*i] - xz[2*i + 1]*y[2*i + 1]
xz[2*i + 1] = tmp*y[2*i + 1] + xz[2*i + 1]*y[2*i]
```

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz*     Pointer to the first element of the first source and destination vector.
- y*     Pointer to the first element of the second source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMul\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorMul\_U8\_U8\_Mod, mlib\_VectorMul\_U8\_U8\_Sat, mlib\_VectorMul\_U8C\_U8C\_Mod, mlib\_VectorMul\_U8C\_U8C\_Sat, mlib\_VectorMul\_S8\_S8\_Mod, mlib\_VectorMul\_S8\_S8\_Sat, mlib\_VectorMul\_S8C\_S8C\_Mod, mlib\_VectorMul\_S8C\_S8C\_Sat, mlib\_VectorMul\_S16\_U8\_Mod, mlib\_VectorMul\_S16\_U8\_Sat, mlib\_VectorMul\_S16\_S8\_Mod, mlib\_VectorMul\_S16\_S8\_Sat, mlib\_VectorMul\_S16\_S16\_Mod, mlib\_VectorMul\_S16\_S16\_Sat, mlib\_VectorMul\_S16C\_U8C\_Mod, mlib\_VectorMul\_S16C\_U8C\_Sat, mlib\_VectorMul\_S16C\_S8C\_Mod, mlib\_VectorMul\_S16C\_S8C\_Sat, mlib\_VectorMul\_S16C\_S16C\_Mod, mlib\_VectorMul\_S16C\_S16C\_Sat, mlib\_VectorMul\_S32\_S16\_Mod, mlib\_VectorMul\_S32\_S16\_Sat, mlib\_VectorMul\_S32\_S32\_Mod, mlib\_VectorMul\_S32\_S32\_Sat, mlib\_VectorMul\_S32C\_S16C\_Mod, mlib\_VectorMul\_S32C\_S16C\_Sat, mlib\_VectorMul\_S32C\_S32C\_Mod, mlib\_VectorMul\_S32C\_S32C\_Sat – vector multiplication

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VectorMul_U8_U8_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_U8_U8_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_U8C_U8C_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_U8C_U8C_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S8_S8_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S8_S8_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S8C_S8C_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S8C_S8C_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16_U8_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16_U8_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16_S8_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
```

```
mllib_status mllib_VectorMul_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_U8C_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_U8C_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_S8C_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_S8C_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32_S16_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32_S16_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32_S32_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32_S32_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32C_S16C_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32C_S16C_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32C_S32C_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mllib_status mllib_VectorMul_S32C_S32C_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
```

**Description** Each of these functions multiplies one vector by another vector.

For real data, the following equation is used:

$$z[i] = x[i] * y[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= x[2*i]*y[2*i] - x[2*i + 1]*y[2*i + 1] \\ z[2*i + 1] &= x[2*i]*y[2*i + 1] + x[2*i + 1]*y[2*i] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorMul\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorNorm\_U8\_Sat, mllib\_VectorNorm\_S8\_Sat, mllib\_VectorNorm\_S16\_Sat, mllib\_VectorNorm\_S32\_Sat – vector norm

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorNorm_U8_Sat(mllib_d64 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorNorm_S8_Sat(mllib_d64 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorNorm_S16_Sat(mllib_d64 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorNorm_S32_Sat(mllib_d64 *z, const mllib_s32 *x,
 mllib_s32 n);
```

**Description** Each of these functions computes the vector normal.

The following equation is used:

$$z[0] = ( \sum_{i=0}^{n-1} x[i]**2 )**0.5$$

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the norm of the vector.
- x*     Pointer to the first element of the source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorReverseByteOrder – reverse byte order of vector

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorReverseByteOrder(void *z, const void *x,
 mllib_s32 n, mllib_s32 s);
```

**Description** The `mllib_VectorReverseByteOrder()` function changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It copies and reverses the byte order of each element of the input vector into the output vector.

**Parameters** The function takes the following arguments:

- z*     Pointer to the output vector.
- x*     Pointer to the input vector.
- n*     Number of elements in the vectors.
- s*     Size of elements in bytes.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorReverseByteOrder\\_Inp\(3MLIB\)](#),  
[mllib\\_VectorReverseByteOrder\\_S16\(3MLIB\)](#),  
[mllib\\_VectorReverseByteOrder\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorReverseByteOrder\_Inp – reverse byte order of vector, in place

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VectorReverseByteOrder_Inp(void *xz,
 mllib_s32 n, mllib_s32 s);
```

**Description** The `mllib_VectorReverseByteOrder_Inp()` function changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It reverses the byte order of each element of the vector, in place.

**Parameters** The function takes the following arguments:

*xz*     Pointer to the input and output vector.

*n*       Number of elements in the vectors.

*s*       Size of elements in bytes.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorReverseByteOrder\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_S16\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VectorReverseByteOrder\_S16, mllib\_VectorReverseByteOrder\_U16, mllib\_VectorReverseByteOrder\_S32, mllib\_VectorReverseByteOrder\_U32, mllib\_VectorReverseByteOrder\_S64, mllib\_VectorReverseByteOrder\_U64, mllib\_VectorReverseByteOrder\_F32, mllib\_VectorReverseByteOrder\_D64 – reverse byte order of vector, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorReverseByteOrder_S16(mllib_s16 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_U16(mllib_u16 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_S32(mllib_s32 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_U32(mllib_u32 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_S64(mllib_s64 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_U64(mllib_u64 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_F32(mllib_f32 *xz,
 mllib_s32 n);

mllib_status mllib_VectorReverseByteOrder_D64(mllib_d64 *xz,
 mllib_s32 n);
```

**Description** Each of these functions changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It reverses the byte order of each element of the vector, in place.

**Parameters** Each of the functions takes the following arguments:

*xz*     Pointer to input and output vector.  
*n*       Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** [mllib\\_VectorReverseByteOrder\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_Inp\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_S16\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorReverseByteOrder\_S16\_S16, mlib\_VectorReverseByteOrder\_U16\_U16, mlib\_VectorReverseByteOrder\_S32\_S32, mlib\_VectorReverseByteOrder\_U32\_U32, mlib\_VectorReverseByteOrder\_S64\_S64, mlib\_VectorReverseByteOrder\_U64\_U64, mlib\_VectorReverseByteOrder\_F32\_F32, mlib\_VectorReverseByteOrder\_D64\_D64 – reverse byte order of vector

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VectorReverseByteOrder_S16_S16(mlib_s16 *z,
 const mlib_s16 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_U16_U16(mlib_u16 *z,
 const mlib_u16 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_S32_S32(mlib_s32 *z,
 const mlib_s32 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_U32_U32(mlib_u32 *z,
 const mlib_u32 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_S64_S64(mlib_s64 *z,
 const mlib_s64 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_U64_U64(mlib_u64 *z,
 const mlib_u64 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_F32_F32(mlib_f32 *z,
 const mlib_f32 *x, mlib_s32 n);

mlib_status mlib_VectorReverseByteOrder_D64_D64(mlib_d64 *z,
 const mlib_d64 *x, mlib_s32 n);
```

**Description** Each of these functions changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It copies and reverses the byte order of each element of the input vector into the output vector.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the output vector.
- x*     Pointer to input vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorReverseByteOrder\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_Inp\(3MLIB\)](#), [mllib\\_VectorReverseByteOrder\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorScale\_U8\_Mod, mllib\_VectorScale\_U8\_Sat, mllib\_VectorScale\_U8C\_Mod, mllib\_VectorScale\_U8C\_Sat, mllib\_VectorScale\_S8\_Mod, mllib\_VectorScale\_S8\_Sat, mllib\_VectorScale\_S8C\_Mod, mllib\_VectorScale\_S8C\_Sat, mllib\_VectorScale\_S16\_Mod, mllib\_VectorScale\_S16\_Sat, mllib\_VectorScale\_S16C\_Mod, mllib\_VectorScale\_S16C\_Sat, mllib\_VectorScale\_S32\_Mod, mllib\_VectorScale\_S32\_Sat, mllib\_VectorScale\_S32C\_Mod, mllib\_VectorScale\_S32C\_Sat – vector linear scaling, in place

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_VectorScale_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);
```

```
mllib_status mllib_VectorScale_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);
```

**Description** Each of these functions performs an in-place scaling of a vector by multiplying by a scalar and adding an offset.

For real data, the following equation is used:

$$xz[i] = a[0]*xz[i] + b[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} tmp &= xz[2*i] \\ xz[2*i] &= a[0]*tmp - a[1]*xz[2*i + 1] + b[0] \\ xz[2*i + 1] &= a[1]*tmp + a[0]*xz[2*i + 1] + b[1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the source and destination vector.
- a* Pointer to the source scaling factor. When the function is used with complex data types, *a*[0] contains the real part of the scaling factor, and *a*[1] contains the imaginary part of the scaling factor.
- b* Pointer to the source offset. When the function is used with complex data types, *b*[0] contains the real part of the offset, and *b*[1] contains the imaginary part of the offset.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorScale\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorScale\_U8\_U8\_Mod, mllib\_VectorScale\_U8\_U8\_Sat, mllib\_VectorScale\_U8C\_U8C\_Mod, mllib\_VectorScale\_U8C\_U8C\_Sat, mllib\_VectorScale\_S8\_S8\_Mod, mllib\_VectorScale\_S8\_S8\_Sat, mllib\_VectorScale\_S8C\_S8C\_Mod, mllib\_VectorScale\_S8C\_S8C\_Sat, mllib\_VectorScale\_S16\_U8\_Mod, mllib\_VectorScale\_S16\_U8\_Sat, mllib\_VectorScale\_S16\_S8\_Mod, mllib\_VectorScale\_S16\_S8\_Sat, mllib\_VectorScale\_S16\_S16\_Mod, mllib\_VectorScale\_S16\_S16\_Sat, mllib\_VectorScale\_S16C\_U8C\_Mod, mllib\_VectorScale\_S16C\_U8C\_Sat, mllib\_VectorScale\_S16C\_S8C\_Mod, mllib\_VectorScale\_S16C\_S8C\_Sat, mllib\_VectorScale\_S16C\_S16C\_Mod, mllib\_VectorScale\_S16C\_S16C\_Sat, mllib\_VectorScale\_S32\_S16\_Mod, mllib\_VectorScale\_S32\_S16\_Sat, mllib\_VectorScale\_S32\_S32\_Mod, mllib\_VectorScale\_S32\_S32\_Sat, mllib\_VectorScale\_S32C\_S16C\_Mod, mllib\_VectorScale\_S32C\_S16C\_Sat, mllib\_VectorScale\_S32C\_S32C\_Mod, mllib\_VectorScale\_S32C\_S32C\_Sat – vector linear scaling

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorScale_U8_U8_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8_U8_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8C_U8C_Mod(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_U8C_U8C_Sat(mllib_u8 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8_S8_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8_S8_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8C_S8C_Mod(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S8C_S8C_Sat(mllib_s8 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_U8_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_U8_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_S8_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);
```

```
mllib_status mllib_VectorScale_S16_S8_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_S16_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16_S16_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_U8C_Mod(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_U8C_Sat(mllib_s16 *z, const mllib_u8 *x,
 const mllib_u8 *a, const mllib_u8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_S8C_Mod(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_S8C_Sat(mllib_s16 *z, const mllib_s8 *x,
 const mllib_s8 *a, const mllib_s8 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_S16C_Mod(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S16C_S16C_Sat(mllib_s16 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_S16_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_S16_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_S32_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32_S32_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32C_S16C_Mod(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32C_S16C_Sat(mllib_s32 *z, const mllib_s16 *x,
 const mllib_s16 *a, const mllib_s16 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32C_S32C_Mod(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);

mllib_status mllib_VectorScale_S32C_S32C_Sat(mllib_s32 *z, const mllib_s32 *x,
 const mllib_s32 *a, const mllib_s32 *b, mllib_s32 n);
```

**Description** Each of these functions scales a vector by multiplying by a scalar and adding an offset.

For real data, the following equation is used:

$$z[i] = a[0]*x[i] + b[0]$$



where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= a[0]*x[2*i] - a[1]*x[2*i + 1] + b[0] \\ z[2*i + 1] &= a[1]*x[2*i] + a[0]*x[2*i + 1] + b[1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the first element of the destination vector.
- x* Pointer to the first element of the source vector.
- a* Pointer to the source scaling factor. When the function is used with complex data types, *a*[0] contains the real part of the scaling factor, and *a*[1] contains the imaginary part of the scaling factor.
- b* Pointer to the source offset. When the function is used with complex data types, *b*[0] contains the real part of the offset, and *b*[1] contains the imaginary part of the offset.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorScale\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorSet\_U8, mllib\_VectorSet\_U8C, mllib\_VectorSet\_S8, mllib\_VectorSet\_S8C, mllib\_VectorSet\_S16, mllib\_VectorSet\_S16C, mllib\_VectorSet\_S32, mllib\_VectorSet\_S32C – set vector to specified value

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorSet_U8(mllib_u8 *z, const mllib_u8 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_U8C(mllib_u8 *z, const mllib_u8 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S8(mllib_s8 *z, const mllib_s8 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S8C(mllib_s8 *z, const mllib_s8 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S16(mllib_s16 *z, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S16C(mllib_s16 *z, const mllib_s16 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S32(mllib_s32 *z, const mllib_s32 *c,
 mllib_s32 n);

mllib_status mllib_VectorSet_S32C(mllib_s32 *z, const mllib_s32 *c,
 mllib_s32 n);
```

**Description** Each of these functions sets a vector to a specified value.

For real data, the following equation is used:

$$z[i] = c[0]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$z[2*i] = c[0]$$

$$z[2*i + 1] = c[1]$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

*z* Pointer to the first element of the destination vector.

*c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scaling factor for the real part, and *c*[1] contains the scaling factor for the imaginary part.

*n* Number of elements in the vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorSplit\_U8\_U8C, mllib\_VectorSplit\_S8\_S8C, mllib\_VectorSplit\_S16\_S16C, mllib\_VectorSplit\_S32\_S32C – vector split

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorSplit_U8_U8C(mllib_u8 *r, mllib_u8 *i,
 const mllib_u8 *x, mllib_s32 n);

mllib_status mllib_VectorSplit_S8_S8C(mllib_s8 *r, mllib_s8 *i,
 const mllib_s8 *x, mllib_s32 n);

mllib_status mllib_VectorSplit_S16_S16C(mllib_s16 *r, mllib_s16 *i,
 const mllib_s16 *x, mllib_s32 n);

mllib_status mllib_VectorSplit_S32_S32C(mllib_s32 *r, mllib_s32 *i,
 const mllib_s32 *x, mllib_s32 n);
```

**Description** Each of these functions splits a complex vector into separate vectors containing the real and imaginary parts.

The following equation is used:

$$r[k] = z[2*k]$$

$$i[k] = z[2*k + 1]$$

where  $k = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- r* Pointer to the first element of the real part.
- i* Pointer to the first element of the imaginary part.
- x* Pointer to the first complex element of the source vector.  $x[2*k]$  contains the real part, and  $x[2*k + 1]$  contains the imaginary part.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VectorMerge\\_U8C\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorSubS\_U8\_Mod, mllib\_VectorSubS\_U8\_Sat, mllib\_VectorSubS\_U8C\_Mod, mllib\_VectorSubS\_U8C\_Sat, mllib\_VectorSubS\_S8\_Mod, mllib\_VectorSubS\_S8\_Sat, mllib\_VectorSubS\_S8C\_Mod, mllib\_VectorSubS\_S8C\_Sat, mllib\_VectorSubS\_S16\_Mod, mllib\_VectorSubS\_S16\_Sat, mllib\_VectorSubS\_S16C\_Mod, mllib\_VectorSubS\_S16C\_Sat, mllib\_VectorSubS\_S32\_Mod, mllib\_VectorSubS\_S32\_Sat, mllib\_VectorSubS\_S32C\_Mod, mllib\_VectorSubS\_S32C\_Sat – vector subtraction from scalar, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorSubS_U8_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8C_Mod(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8C_Sat(mllib_u8 *xz,
 const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8C_Mod(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8C_Sat(mllib_s8 *xz,
 const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16C_Mod(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16C_Sat(mllib_s16 *xz,
 const mllib_s16 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S32_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S32_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S32C_Mod(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);
```

```
mllib_status mllib_VectorSubS_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *c, mllib_s32 n);
```

**Description** Each of these functions performs an in-place subtraction of a vector from a scalar.

For real data, the following equation is used:

$$xz[i] = c[0] - xz[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} xz[2*i] &= c[0] - xz[2*i] \\ xz[2*i + 1] &= c[1] - xz[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

**Parameters** Each of the functions takes the following arguments:

- xz* Pointer to the first element of the source and destination vector.
- c* Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorSubS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorSubS\_U8\_U8\_Mod, mllib\_VectorSubS\_U8\_U8\_Sat, mllib\_VectorSubS\_U8C\_U8C\_Mod, mllib\_VectorSubS\_U8C\_U8C\_Sat, mllib\_VectorSubS\_S8\_S8\_Mod, mllib\_VectorSubS\_S8\_S8\_Sat, mllib\_VectorSubS\_S8C\_S8C\_Mod, mllib\_VectorSubS\_S8C\_S8C\_Sat, mllib\_VectorSubS\_S16\_U8\_Mod, mllib\_VectorSubS\_S16\_U8\_Sat, mllib\_VectorSubS\_S16\_S8\_Mod, mllib\_VectorSubS\_S16\_S8\_Sat, mllib\_VectorSubS\_S16\_S16\_Mod, mllib\_VectorSubS\_S16\_S16\_Sat, mllib\_VectorSubS\_S16C\_U8C\_Mod, mllib\_VectorSubS\_S16C\_U8C\_Sat, mllib\_VectorSubS\_S16C\_S8C\_Mod, mllib\_VectorSubS\_S16C\_S8C\_Sat, mllib\_VectorSubS\_S16C\_S16C\_Mod, mllib\_VectorSubS\_S16C\_S16C\_Sat, mllib\_VectorSubS\_S32\_S16\_Mod, mllib\_VectorSubS\_S32\_S16\_Sat, mllib\_VectorSubS\_S32\_S32\_Mod, mllib\_VectorSubS\_S32\_S32\_Sat, mllib\_VectorSubS\_S32C\_S16C\_Mod, mllib\_VectorSubS\_S32C\_S16C\_Sat, mllib\_VectorSubS\_S32C\_S32C\_Mod, mllib\_VectorSubS\_S32C\_S32C\_Sat – vector subtraction from scalar

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VectorSubS_U8_U8_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8_U8_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8C_U8C_Mod(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_U8C_U8C_Sat(mllib_u8 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8_S8_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8_S8_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8C_S8C_Mod(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S8C_S8C_Sat(mllib_s8 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16_U8_Mod(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16_U8_Sat(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *c, mllib_s32 n);

mllib_status mllib_VectorSubS_S16_S8_Mod(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *c, mllib_s32 n);
```



```

mllib_status mllib_VectorSubS_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16_S16_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16_S16_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_U8C_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_U8C_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_S8C_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_S8C_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_S16C_Mod(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S16C_S16C_Sat(mlib_s16 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32_S16_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32_S16_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32_S32_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32_S32_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32C_S16C_Mod(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32C_S16C_Sat(mlib_s32 *z,
 const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32C_S32C_Mod(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

mllib_status mllib_VectorSubS_S32C_S32C_Sat(mlib_s32 *z,
 const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

```

**Description** Each of these functions subtracts a vector from a scalar.

For real data, the following equation is used:

$$z[i] = c[0] - x[i]$$

where  $i = 0, 1, \dots, (n - 1)$ .

For complex data, the following equation is used:

$$\begin{aligned} z[2*i] &= c[0] - x[2*i] \\ z[2*i + 1] &= c[1] - x[2*i + 1] \end{aligned}$$

where  $i = 0, 1, \dots, (n - 1)$ .

- Parameters** Each of the functions takes the following arguments:
- z*     Pointer to the first element of the destination vector.
  - x*     Pointer to the first element of the source vector.
  - c*     Pointer to the source scalar. When the function is used with complex data types, *c*[0] contains the scalar for the real part, and *c*[1] contains the scalar for the imaginary part.
  - n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorSubS\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorSub\_U8\_Mod, mlib\_VectorSub\_U8\_Sat, mlib\_VectorSub\_U8C\_Mod, mlib\_VectorSub\_U8C\_Sat, mlib\_VectorSub\_S8\_Mod, mlib\_VectorSub\_S8\_Sat, mlib\_VectorSub\_S8C\_Mod, mlib\_VectorSub\_S8C\_Sat, mlib\_VectorSub\_S16\_Mod, mlib\_VectorSub\_S16\_Sat, mlib\_VectorSub\_S16C\_Mod, mlib\_VectorSub\_S16C\_Sat, mlib\_VectorSub\_S32\_Mod, mlib\_VectorSub\_S32\_Sat, mlib\_VectorSub\_S32C\_Mod, mlib\_VectorSub\_S32C\_Sat – vector subtraction, in place

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mlib.h>`

```

mlib_status mlib_VectorSub_U8_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8C_Mod(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8C_Sat(mlib_u8 *xz,
 const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8C_Mod(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8C_Sat(mlib_s8 *xz,
 const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16C_Mod(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16C_Sat(mlib_s16 *xz,
 const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S32_Mod(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S32_Sat(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S32C_Mod(mlib_s32 *xz,
 const mlib_s32 *y, mlib_s32 n);

```

```
mllib_status mllib_VectorSub_S32C_Sat(mllib_s32 *xz,
 const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions performs an in-place subtraction of a vector from another vector.

It uses the following equation:

$$xz[i] = xz[i] - y[i]$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- xz*     Pointer to the first element of the first source and destination vector.
- y*     Pointer to the first element of the second source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VectorSub\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VectorSub\_U8\_U8\_Mod, mlib\_VectorSub\_U8\_U8\_Sat,  
 mlib\_VectorSub\_U8C\_U8C\_Mod, mlib\_VectorSub\_U8C\_U8C\_Sat,  
 mlib\_VectorSub\_S8\_S8\_Mod, mlib\_VectorSub\_S8\_S8\_Sat,  
 mlib\_VectorSub\_S8C\_S8C\_Mod, mlib\_VectorSub\_S8C\_S8C\_Sat,  
 mlib\_VectorSub\_S16\_U8\_Mod, mlib\_VectorSub\_S16\_U8\_Sat,  
 mlib\_VectorSub\_S16\_S8\_Mod, mlib\_VectorSub\_S16\_S8\_Sat,  
 mlib\_VectorSub\_S16\_S16\_Mod, mlib\_VectorSub\_S16\_S16\_Sat,  
 mlib\_VectorSub\_S16C\_U8C\_Mod, mlib\_VectorSub\_S16C\_U8C\_Sat,  
 mlib\_VectorSub\_S16C\_S8C\_Mod, mlib\_VectorSub\_S16C\_S8C\_Sat,  
 mlib\_VectorSub\_S16C\_S16C\_Mod, mlib\_VectorSub\_S16C\_S16C\_Sat,  
 mlib\_VectorSub\_S32\_S16\_Mod, mlib\_VectorSub\_S32\_S16\_Sat,  
 mlib\_VectorSub\_S32\_S32\_Mod, mlib\_VectorSub\_S32\_S32\_Sat,  
 mlib\_VectorSub\_S32C\_S16C\_Mod, mlib\_VectorSub\_S32C\_S16C\_Sat,  
 mlib\_VectorSub\_S32C\_S32C\_Mod, mlib\_VectorSub\_S32C\_S32C\_Sat – vector subtraction

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
 #include <mlib.h>

```

mlib_status mlib_VectorSub_U8_U8_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8_U8_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8C_U8C_Mod(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_U8C_U8C_Sat(mlib_u8 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8_S8_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8_S8_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8C_S8C_Mod(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S8C_S8C_Sat(mlib_s8 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_U8_Mod(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_U8_Sat(mlib_s16 *z,
 const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_S8_Mod(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorSub_S16_S8_Sat(mlib_s16 *z,
 const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

```

```
mllib_status mllib_VectorSub_S16_S16_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16_S16_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_U8C_Mod(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_U8C_Sat(mllib_s16 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_S8C_Mod(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_S8C_Sat(mllib_s16 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_S16C_Mod(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S16C_S16C_Sat(mllib_s16 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32_S16_Mod(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32_S16_Sat(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32_S32_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32_S32_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32C_S16C_Mod(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32C_S16C_Sat(mllib_s32 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32C_S32C_Mod(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);

mllib_status mllib_VectorSub_S32C_S32C_Sat(mllib_s32 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions subtracts one vector from another vector.

It uses the following equation:

$$z[i] = x[i] - y[i]$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the first element of the destination vector.
- x*     Pointer to the first element of the first source vector.
- y*     Pointer to the first element of the second source vector.
- n*     Number of elements in the vectors.

**Return Values**   Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**   See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**   [mllib\\_VectorSub\\_U8\\_U8\\_Mod\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VectorSumAbsDiff\_U8\_Sat, mllib\_VectorSumAbsDiff\_S8\_Sat, mllib\_VectorSumAbsDiff\_S16\_Sat, mllib\_VectorSumAbsDiff\_S32\_Sat – sum of the absolute values of the differences of two vectors

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorSumAbsDiff_U8_Sat(mllib_d64 *z,
 const mllib_u8 *x, const mllib_u8 *y, mllib_s32 n);

mllib_status mllib_VectorSumAbsDiff_S8_Sat(mllib_d64 *z,
 const mllib_s8 *x, const mllib_s8 *y, mllib_s32 n);

mllib_status mllib_VectorSumAbsDiff_S16_Sat(mllib_d64 *z,
 const mllib_s16 *x, const mllib_s16 *y, mllib_s32 n);

mllib_status mllib_VectorSumAbsDiff_S32_Sat(mllib_d64 *z,
 const mllib_s32 *x, const mllib_s32 *y, mllib_s32 n);
```

**Description** Each of these functions computes the sum of the absolute values of the differences of two vectors.

The following equation is used:

$$z[0] = \sum_{i=0}^{n-1} |x[i] - y[i]|$$

**Parameters** Each of the functions takes the following arguments:

- z* Pointer to the sum of the absolute differences between two vectors.
- x* Pointer to the first element of the first source vector.
- y* Pointer to the first element of the second source vector.
- n* Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)



**Name** mllib\_VectorSumAbs\_U8\_Sat, mllib\_VectorSumAbs\_S8\_Sat, mllib\_VectorSumAbs\_S16\_Sat, mllib\_VectorSumAbs\_S32\_Sat – sum of the absolute values of a vector

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorSumAbs_U8_Sat(mllib_d64 *z, const mllib_u8 *x,
 mllib_s32 n);

mllib_status mllib_VectorSumAbs_S8_Sat(mllib_d64 *z, const mllib_s8 *x,
 mllib_s32 n);

mllib_status mllib_VectorSumAbs_S16_Sat(mllib_d64 *z, const mllib_s16 *x,
 mllib_s32 n);

mllib_status mllib_VectorSumAbs_S32_Sat(mllib_d64 *z, const mllib_s32 *x,
 mllib_s32 n);
```

**Description** Each of these functions computes the sum of the absolute values of a vector.  
The following equation is used:

$$z[0] = \sum_{i=0}^{n-1} |x[i]|$$

**Parameters** Each of the functions takes the following arguments:

- z*     Pointer to the sum of the absolute values of the vector.
- x*     Pointer to the first element of the first source vector.
- n*     Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VectorZero\_U8, mllib\_VectorZero\_U8C, mllib\_VectorZero\_S8, mllib\_VectorZero\_S8C, mllib\_VectorZero\_S16, mllib\_VectorZero\_S16C, mllib\_VectorZero\_S32, mllib\_VectorZero\_S32C – initialize vector to zero

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VectorZero_U8(mllib_u8 *z, mllib_s32 n);
mllib_status mllib_VectorZero_U8C(mllib_u8 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S8(mllib_s8 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S8C(mllib_s8 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S16(mllib_s16 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S16C(mllib_s16 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S32(mllib_s32 *z, mllib_s32 n);
mllib_status mllib_VectorZero_S32C(mllib_s32 *z, mllib_s32 n);
```

**Description** Each of these functions initializes a vector to zero.

The following equation is used:

$$z[i] = 0$$

where  $i = 0, 1, \dots, (n - 1)$  for real data;  $i = 0, 1, \dots, (2*n - 1)$  for complex data.

**Parameters** Each of the functions takes the following arguments:

- $z$  Pointer to the first element of the destination vector.
- $n$  Number of elements in the vector.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_version – return a version string

**Synopsis**

```
cc [flag...] file... -lmLib [library...]
#include <mLib.h>
```

```
char *mLib_version(void);
```

**Description** The `mLib_version()` function returns a string about the version of the library being used.

This function returns a string in the following format:

*lib\_name:version:build\_date:target\_isa*

The *lib\_name* is `mediaLib`. The *version* consists of four digits. The first two digits of the version are the major version. The third digit is the minor version, and the fourth digit is the micro version. The *build\_date* is in the `yyyymmdd` format. The *target\_isa* is the value used for the `-xarch=a` flag of the compiler when the library was built. For example, the following version string corresponds to a library in `mediaLib` version 2.1.0, which was built on 11/01/2001 and for the `sparcv8plus+vis` architecture.

`mediaLib:0210:20011101:v8plusa`

**Parameters** The function takes no argument.

**Return Values** The function returns a pointer to a string of characters.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VideoAddBlock\_U8\_S16 – adds motion-compensated 8x8 block to the current block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoAddBlock_U8_S16(mllib_u8 *curr_block,
 const mllib_s16 *mc_block, mllib_s32 stride);
```

**Description** The `mllib_VideoAddBlock_U8_S16()` function performs additions of prediction and coefficient data. In other words, the function adds a motion-compensated 8x8 block to the current block. The stride applies to the current block.

**Parameters** The function takes the following arguments:

- curr\_block*      Pointer to the current block. *curr\_block* must be 8-byte aligned.
- mc\_block*        Pointer to an 8x8 motion-compensated block (prediction data). *mc\_block* must be 8-byte aligned.
- stride*           Stride, in bytes, between adjacent rows in the current block. *stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoCopyRef_S16_U8(3MLIB)`, `mllib_VideoCopyRef_S16_U8_16x16(3MLIB)`,  
`mllib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mllib_VideoH2630verlappedMC_S16_U8(3MLIB)`,  
`mllib_VideoH2630verlappedMC_U8_U8(3MLIB)`, `mllib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpX_S16_U8(3MLIB)`,  
`mllib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpX_U8_U8(3MLIB)`,  
`mllib_VideoInterpXY_S16_U8(3MLIB)`, `mllib_VideoInterpXY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpXY_U8_U8(3MLIB)`, `mllib_VideoInterpXY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_S16_U8(3MLIB)`, `mllib_VideoInterpY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_U8_U8(3MLIB)`, `mllib_VideoInterpY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoP64Decimate_U8_U8(3MLIB)`, `mllib_VideoP64Loop_S16_U8(3MLIB)`,  
`mllib_VideoP64Loop_U8_U8(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_VideoColorABGR2JFIFYCC420 – ABGR to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorABGR2JFIFYCC420(mllib_u8 *y0,
 mllib_u8 *y1, mllib_u8 *cb, mllib_u8 *cr, const mllib_u8 *abgr0,
 const mllib_u8 *abgr1, mllib_s32 n);
```

**Description** The `mllib_VideoColorABGR2JFIFYCC420()` function performs color space conversion from ABGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.

*y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.

*cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.

*cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.

*abgr0* Pointer to upper source ABGR multi-component row. *abgr0* must be 8-byte aligned.

*abgr1* Pointer to lower source ABGR multi-component row. *abgr1* must be 8-byte aligned.

*n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the ABGR multi-component row must be 4\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC422\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorABGR2JFIFYCC422 – ABGR to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorABGR2JFIFYCC422(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *abgr, mllib_s32 n);
```

**Description** The `mllib_VideoColorABGR2JFIFYCC422()` function performs color space conversion from ABGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- y* Pointer to destination Y component row. *y* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- abgr* Pointer to source ABGR multi-component row. *abgr* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the ABGR multi-component row must be 4\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC422\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorABGR2JFIFYCC444 – ABGR to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorABGR2JFIFYCC444(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *abgr, mllib_s32 n);
```

**Description** The `mllib_VideoColorABGR2JFIFYCC444()` function performs color space conversion from and ABGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128 \end{aligned}$$

**Parameters** The function takes the following arguments:

*y* Pointer to destination Y component row. *y* must be 8-byte aligned.

*cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.

*cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.

*abgr* Pointer to source ABGR multi-component row. *abgr* must be 8-byte aligned.

*n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the ABGR multi-component row must be 4\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorARGB2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)





## REFERENCE

### Multimedia Library Functions - Part 7

**Name** mllib\_VideoColorABGR2RGB – color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorABGR2RGB(mllib_u8 *rgb, const mllib_u8 *abgr,
 mllib_s32 n);
```

**Description** The `mllib_VideoColorABGR2RGB()` function performs ABGR to RGB color order conversion.

**Parameters** The function takes the following arguments:

- rgb*        Pointer to RGB row.
- abgr*      Pointer to ABGR row.
- n*         Number of pixels.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorARGB2RGB\(3MLIB\)](#), [mllib\\_VideoColorRGB2ABGR\(3MLIB\)](#), [mllib\\_VideoColorRGB2ARGB\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorABGRInt\_to\_ARGBInt – convert ABGR interleaved to ARGB

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorABGRInt_to_ARGBInt(mllib_u32 *ARGB,
 const mllib_u32 *ABGR, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The ABGR pixel stream is broken apart and recombined into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions *w* and *h*. Within each 32-bit input word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), R (bits 23-16), G (bits 15-8), and B (bits 7-0).

**Parameters** The function takes the following arguments:

- ARGB*     Pointer to output buffer.
- ABGR*     Pointer to input buffer.
- w*         Image width in pixels.
- h*         Image height in lines.
- dlb*       Linebytes for output buffer.
- slb*       Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorRGBaint\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorBGRaint\\_to\\_ABGRInt\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorARGB2JFIFYCC420 – ARGB to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorARGB2JFIFYCC420(mllib_u8 *y0,
 mllib_u8 *y1, mllib_u8 *cb, mllib_u8 *cr, const mllib_u8 *argb0,
 const mllib_u8 *argb1, mllib_s32 n);
```

**Description** The `mllib_VideoColorARGB2JFIFYCC420()` function performs color space conversion from ARGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.
- y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- argb0* Pointer to upper source ARGB multi-component row. *argb0* must be 8-byte aligned.
- argb1* Pointer to lower source ARGB multi-component row. *argb1* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the ARGB multi-component row must be 4\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorABGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC422\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorARGB2JFIFYCC422 – ARGB to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorARGB2JFIFYCC422(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *argb, mllib_s32 n);
```

**Description** The `mllib_VideoColorARGB2JFIFYCC422()` function performs color space conversion from ARGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*y*            Pointer to destination Y component row. *y* must be 8-byte aligned.  
*cb*           Pointer to destination Cb component row. *cb* must be 8-byte aligned.  
*cr*           Pointer to destination Cr component row. *cr* must be 8-byte aligned.  
*argb*        Pointer to source ARGB multi-component row. *argb* must be 8-byte aligned.  
*n*            Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the ARGB multi-component row must be 4\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorABGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC422\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorARGB2JFIFYCC444 – ARGB to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorARGB2JFIFYCC444(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *argb, mllib_s32 n);
```

**Description** The `mllib_VideoColorARGB2JFIFYCC444()` function performs color space conversion from ARGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128 \end{aligned}$$

**Parameters** The function takes the following arguments:

- y* Pointer to destination Y component row. *y* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- argb* Pointer to source ARGB multi-component row. *rgb* must be 8-byte aligned.
- n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the ARGB multi-component row must be 4\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorARGB2RGB – color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorARGB2RGB(mllib_u8 *rgb,
 const mllib_u8 *argb, mllib_s32 n);
```

**Description** The `mllib_VideoColorARGB2RGB()` function performs ARGB to RGB color order conversion.

**Parameters** The function takes the following arguments:

- rgb* Pointer to RGB row.
- argb* Pointer to ARGB row.
- n* Number of pixels.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2RGB\(3MLIB\)](#), [mllib\\_VideoColorRGB2ABGR\(3MLIB\)](#), [mllib\\_VideoColorRGB2ARGB\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorBGR2JFIFYCC420 – BGR to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorBGR2JFIFYCC420(mllib_u8 *y0, mllib_u8 *y1,
 mllib_u8 *cb, mllib_u8 *cr, const mllib_u8 *bgr0, const mllib_u8 *bgr1,
 mllib_s32 n);
```

**Description** The `mllib_VideoColorBGR2JFIFYCC420()` function performs color space conversion from BGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.
- y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- bgr0* Pointer to upper source BGR multi-component row. *bgr0* must be 8-byte aligned.
- bgr1* Pointer to lower source BGR multi-component row. *bgr1* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the BGR multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorBGR2JFIFYCC422 – BGR to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorBGR2JFIFYCC422(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *bgr, mllib_s32 n);
```

**Description** The `mllib_VideoColorBGR2JFIFYCC422()` function performs color space conversion from BGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*y*        Pointer to destination Y component row. *y* must be 8-byte aligned.  
*cb*       Pointer to destination Cb component row. *cb* must be 8-byte aligned.  
*cr*       Pointer to destination Cr component row. *cr* must be 8-byte aligned.  
*bgr*      Pointer to source BGR multi-component row. *bgr* must be 8-byte aligned.  
*n*        Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be  $n/2$ . The length of the BGR multi-component row must be  $3*n$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorBGR2JFIFYCC444 – BGR to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorBGR2JFIFYCC444(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *bgr, mllib_s32 n);
```

**Description** The `mllib_VideoColorBGR2JFIFYCC444()` function performs color space conversion from BGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128 \end{aligned}$$

**Parameters** The function takes the following arguments:

- y* Pointer to destination Y component row. *y* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- bgr* Pointer to source BGR multi-component row. *bgr* must be 8-byte aligned.
- n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the BGR multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorBGR2JFIFYCC444\_S16 – BGR to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorBGR2JFIFYCC444_S16(mllib_s16 *y, mllib_s16 *cb,
 mllib_s16 *cr, const mllib_s16 *bgr, mllib_s32 n);
```

**Description** The `mllib_VideoColorBGR2JFIFYCC444_S16()` function performs color space conversion from BGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

Both the input BGR components and the output YCbCr components are supposed to be in the range of [0, 4095].

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 2048 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 2048 \end{aligned}$$

**Parameters** The function takes the following arguments:

*y*        Pointer to destination Y component row. *y* must be 8-byte aligned.

*cb*       Pointer to destination Cb component row. *cb* must be 8-byte aligned.

*cr*       Pointer to destination Cr component row. *cr* must be 8-byte aligned.

*bgr*      Pointer to source BGR multi-component row. *bgr* must be 8-byte aligned.

*n*        Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the BGR multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorBGR2JFIFYCC444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorBGRAint\_to\_ABGRint – convert BGRA interleaved to ABGR

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorBGRAint_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u32 *BGRA, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The BGRA pixel stream is broken apart and recombined into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions *w* and *h*. Within each 32-bit input word, the component ordering is B (bits 31-24), G (bits 23-16), R (bits 15-8), and A (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

**Parameters** The function takes the following arguments:

- ABGR*     Pointer to output buffer.
- BGRA*     Pointer to input buffer.
- w*         Image width in pixels.
- h*         Image height in lines.
- dlb*       Linebytes for output buffer.
- slb*       Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGRint\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorRGBAint\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorBGRint\_to\_ABGRint – convert BGR interleaved to ABGR interleaved

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_VideoColorBGRint_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *BGR, const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** The interleaved BGR stream, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 24-bit input pixel, the component ordering is B (bits 23-16), G (bits 15-8), and R (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

**Parameters** The function takes the following arguments:

*ABGR*      Pointer to output buffer.

*BGR*        Pointer to input buffer.

*A\_array*    Array of alpha values.

*A\_const*    Constant alpha value.

*w*           Image width in pixels.

*h*           Image height in lines.

*dlb*        Linebytes for output buffer.

*slb*        Linebytes for input buffer.

*alb*        Linebytes for alpha buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorRGBseq_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorRGBInt_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorRGBXInt_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorRGBXInt_to_ARGBInt(3MLIB)`,  
`mllib_VideoColorXRGBInt_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorXRGBInt_to_ARGBInt(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoColorBlendABGR, mllib\_VideoColorBlendABGR\_ResetAlpha – image blend

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorBlendABGR(mllib_u32 *dst,
 const mllib_u32 *src1, const mllib_u32 *src2,
 mllib_s32 src1_w, mllib_s32 src1_h,
 mllib_s32 src2_w, mllib_s32 src2_h,
 mllib_s32 src2_x, mllib_s32 src2_y,
 mllib_s32 dst_lb, mllib_s32 src1_lb,
 mllib_s32 src2_lb, mllib_blend src1_blend,
 mllib_blend src2_blend);

void mllib_VideoColorBlendABGR_ResetAlpha(mllib_u32 *dst,
 const mllib_u32 *src1, const mllib_u32 *src2,
 mllib_s32 src1_w, mllib_s32 src1_h,
 mllib_s32 src2_w, mllib_s32 src2_h,
 mllib_s32 src2_x, mllib_s32 src2_y,
 mllib_s32 dst_lb, mllib_s32 src1_lb,
 mllib_s32 src2_lb, mllib_blend src1_blend,
 mllib_blend src2_blend);
```

**Description** The functions use the following equation for blending images:

$$dst = (src1 * src1\_blend) + (src2 * src2\_blend)$$

The two multi-banded source images (src1 and src2) are blended together and stored in the destination image (dst). The image buffers pointed to by dst, src1, and src2 contain 4-banded ABGR images, 8 bits per component. src1\_w and src1\_h are the dimensions of the src1 input buffer. src2\_w and src2\_h are the dimensions of the src2 input buffer. The output buffer must be at least as large as the src1 input buffer. src2\_x and src2\_y are the offset of the src2 input buffer relative to src1. Where pixels in src2 overlap pixels in src1, the pixels are blended. Pixels in src1 which are outside of src2 are copied into dst. Pixels in the dst image outside of src1 are left unchanged. src1\_blend specifies the blend function to be applied to the pixels of src1 image and src2\_blend specifies the blend function to be applied to the pixels of src2.

Possible blend functions are:

```
MLIB_BLEND_ZERO
MLIB_BLEND_ONE
MLIB_BLEND_SRC_ALPHA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA
MLIB_BLEND_DST_ALPHA
MLIB_BLEND_ONE_MINUS_DST_ALPHA
```

MLIB\_BLEND\_SRC\_ALPHA is the alpha component of image src2 scaled to the range 0.0 to 1.0. MLIB\_BLEND\_DST\_ALPHA is the alpha component of image src1 scaled to the range 0.0 to 1.0. All pixel components are treated as unsigned 8-bit quantities and the output pixel component values are clamped to the range 0 to 255.

For the `mllib_VideoColorBlendABGR_ResetAlpha()` function, the alpha value of every pixel in destination image is set to 0 after blending is complete.

**Parameters** Each of the functions takes the following arguments:

- dst* Pointer to output image.
- src1* Pointer to 1st input image.
- src2* Pointer to 2nd input image.
- src1\_w* src1 image width in pixels.
- src1\_h* src1 image height in rows.
- src2\_w* src2 image width in pixels.
- src2\_h* src2 image height in rows.
- src2\_x* src2 horizontal displacement (in pixels), relative to the upper-left corner of src1.
- src2\_y* src2 vertical displacement (in rows), relative to the upper-left corner of src1.
- dst\_lb* Linebytes for output image.
- src1\_lb* Linebytes for 1st input image.
- src2\_lb* Linebytes for 2nd input image.
- src1\_blend* Blend function for src1 image.
- src2\_blend* Blend function for src2 image.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBlendABGR\\_Inp\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_VideoColorBlendABGR\_Inp, mlib\_VideoColorBlendABGR\_ResetAlpha\_Inp – in-place image blend

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
void mlib_VideoColorBlendABGR_Inp(mlib_u32 *src1dst,
 const mlib_u32 *src2, mlib_s32 src1dst_w,
 mlib_s32 src1dst_h, mlib_s32 src2_w,
 mlib_s32 src2_h, mlib_s32 src2_x,
 mlib_s32 src2_y, mlib_s32 src1dst_lb,
 mlib_s32 src2_lb, mlib_blend src1dst_blend,
 mlib_blend src2_blend);

void mlib_VideoColorBlendABGR_ResetAlpha_Inp(mlib_u32 *src1dst,
 const mlib_u32 *src2, mlib_s32 src1dst_w,
 mlib_s32 src1dst_h, mlib_s32 src2_w,
 mlib_s32 src2_h, mlib_s32 src2_x,
 mlib_s32 src2_y, mlib_s32 src1dst_lb,
 mlib_s32 src2_lb, mlib_blend src1dst_blend,
 mlib_blend src2_blend);
```

**Description** The functions use the following equation for blending images:

$$\text{src1dst} = (\text{src1dst} * \text{src1dst\_blend}) + (\text{src2} * \text{src2\_blend})$$

The two multi-banded source images (src1dst and src2) are blended together and the result is stored in src1dst. src1dst\_blend specifies the blend function to be applied to the src1dst image and src2\_blend specifies the blend function to be applied to the src2 image. src2\_x and src2\_y specify position of src2 relative to the upper-left corner of src1dst. src2 is clipped to the boundaries of src1dst, if needed.

Possible blend functions are:

```
MLIB_BLEND_ZERO
MLIB_BLEND_ONE
MLIB_BLEND_SRC_ALPHA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA
MLIB_BLEND_DST_ALPHA
MLIB_BLEND_ONE_MINUS_DST_ALPHA
```

MLIB\_BLEND\_DST\_ALPHA is the alpha band of image src1 scaled to the range 0 to 1. MLIB\_BLEND\_SRC\_ALPHA is the alpha band of image src2 scaled to the range 0 to 1. The output pixel bands are clamped to the range 0 to 255.

For the mlib\_VideoColorBlendABGR\_ResetAlpha\_Inp() function, the alpha value of every pixel in destination image is set to 0 after blending is complete.

**Parameters** Each of the functions takes the following arguments:

- src1dst* Pointer to 1st input image (also dest. image).
- src2* Pointer to 2nd input image.
- src1dst\_w* src1dst image width in pixels.
- src1dst\_h* src1dst image height in rows.
- src2\_w* src2 image width in pixels.
- src2\_h* src2 image height in rows.
- src2\_x* src2 horizontal displacement (in pixels), relative to the upper-left corner of src1dst.
- src2\_y* src2 vertical displacement (in rows), relative to the upper-left corner of src1dst.
- src1dst\_lb* Linebytes for src1dst image.
- src2\_lb* Linebytes for src2 image.
- src1dst\_blend* Blend function for src1dst image.
- src2\_blend* Blend function for src2 image.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorBlendABGR\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorCMYK2JFIFYCCK444 – CMYK to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorCMYK2JFIFYCCK444(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, mllib_u8 *k, const mllib_u8 *cmyk, mllib_s32 n);
```

**Description** The `mllib_VideoColorCMYK2JFIFYCCK444()` function performs color space conversion from CMYK to YCbCrK when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} R &= (255 - C) \\ G &= (255 - M) \\ B &= (255 - Y) \\ Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128 \\ K &= K \end{aligned}$$

**Parameters** The function takes the following arguments:

*y* Pointer to destination Y component row. *y* must be 8-byte aligned.

*cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.

*cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.

*k* Pointer to destination K component row. *k* must be 8-byte aligned.

*cmyk* Pointer to source CMYK multi-component row. *cmyk* must be 8-byte aligned.

*n* Length of Y,Cb,Cr, and K component rows. The length of the CMYK multi-component row must be 4\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCCK2CMYK444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2ABGR444 – JFIF YCbCr to ABGR color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorJFIFYCC2ABGR444(mllib_u8 *abgr,
 const mllib_u8 *y, const mllib_u8 *cb,
 const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2ABGR444()` function performs color space conversion from YCbCr to ABGR when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} A &= 0xFF \\ R &= Y + 1.40200 * (Cr - 128) \\ G &= Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128) \\ B &= Y + 1.77200 * (Cb - 128) \end{aligned}$$

**Parameters** The function takes the following arguments:

- abgr* Pointer to destination ABGR multi-component row. *abgr* must be 8-byte aligned.
- y* Pointer to source Y component row. *y* must be 8-byte aligned.
- cb* Pointer to source Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to source Cr component row. *cr* must be 8-byte aligned.
- n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the ABGR multi-component row must be 4\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2ARGB444 – JFIF YCbCr to ARGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorJFIFYCC2ARGB444(mllib_u8 *argb,
 const mllib_u8 *y, const mllib_u8 *cb,
 const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2ARGB444()` function performs color space conversion from YCbCr to ARGB when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} A &= 0xFF \\ R &= Y + 1.40200 * (Cr - 128) \\ G &= Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128) \\ B &= Y + 1.77200 * (Cb - 128) \end{aligned}$$

**Parameters** The function takes the following arguments:

*argb* Pointer to destination ARGB multi-component row. *argb* must be 8-byte aligned.

*y* Pointer to source Y component row. *y* must be 8-byte aligned.

*cb* Pointer to source Cb component row. *cb* must be 8-byte aligned.

*cr* Pointer to source Cr component row. *cr* must be 8-byte aligned.

*n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the ARGB multi-component row must be 4\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2RGB420 – JFIF YCbCr to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorJFIFYCC2RGB420(mllib_u8 *rgb0,
 mllib_u8 *rgb1, const mllib_u8 *y0,
 const mllib_u8 *y1, const mllib_u8 *cb0,
 const mllib_u8 *cr0, const mllib_u8 *cb1,
 const mllib_u8 *cr1, const mllib_u8 *cb2,
 const mllib_u8 *cr2, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB420()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- rgb0* Pointer to upper destination RGB multi-component row. *rgb0* must be 8-byte aligned.
- rgb1* Pointer to lower destination RGB multi-component row. *rgb1* must be 8-byte aligned.
- y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.
- y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.
- cb0* Pointer to source upper Cb component row. *cb0* must be 8-byte aligned.
- cr0* Pointer to source upper Cr component row. *cr0* must be 8-byte aligned.
- cb1* Pointer to source middle Cb component row. *cb1* must be 8-byte aligned.
- cr1* Pointer to source middle Cr component row. *cr1* must be 8-byte aligned.
- cb2* Pointer to source lower Cb component row. *cb2* must be 8-byte aligned.
- cr2* Pointer to source lower Cr component row. *cr2* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be  $n/2$ . The length of the RGB multi-component row must be  $3*n$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB)`,  
`mlib_VideoColorJFIFYCC2RGB422(3MLIB)`,  
`mlib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB), attributes(5)`

**Name** mllib\_VideoColorJFIFYCC2RGB420\_Nearest – JFIF YCbCr to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorJFIFYCC2RGB420_Nearest(mllib_u8 *rgb0,
 mllib_u8 *rgb1, const mllib_u8 *y0,
 const mllib_u8 *y1, const mllib_u8 *cb,
 const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB420_Nearest()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- rgb0* Pointer to upper destination RGB multi-component row. *rgb0* must be 8-byte aligned.
- rgb1* Pointer to lower destination RGB multi-component row. *rgb1* must be 8-byte aligned.
- y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.
- y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.
- cb* Pointer to source Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to source Cr component row. *cr* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be  $n/2$ . The length of the RGB multi-component row must be  $3*n$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2RGB420\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB422\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB422\\_Nearest\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorJFIFYCC2RGB422 – JFIF YCbCr to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorJFIFYCC2RGB422(mllib_u8 *rgb,
 const mllib_u8 *y, const mllib_u8 *cb,
 const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB422()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*rgb*     Pointer to destination RGB multi-component row. *rgb* must be 8-byte aligned.

*y*       Pointer to destination Y component row. *y* must be 8-byte aligned.

*cb*      Pointer to source Cb component row. *cb* must be 8-byte aligned.

*cr*      Pointer to source Cr component row. *cr* must be 8-byte aligned.

*n*       Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be  $n/2$ . The length of the RGB multi-component row must be  $3*n$ .

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2RGB420\(3MLIB\)](#),  
[mllib\\_VideoColorJFIFYCC2RGB420\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoColorJFIFYCC2RGB422\\_Nearest\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2RGB422\_Nearest – JFIF YCbCr to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorJFIFYCC2RGB422_Nearest(mllib_u8 *rgb,
 const mllib_u8 *y, const mllib_u8 *cb, const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB422_Nearest()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- rgb* Pointer to destination RGB multi-component row. *rgb* must be 8-byte aligned.
- y* Pointer to destination Y component row. *y* must be 8-byte aligned.
- cb* Pointer to source Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to source Cr component row. *cr* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2RGB420\(3MLIB\)](#),  
[mllib\\_VideoColorJFIFYCC2RGB420\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoColorJFIFYCC2RGB422\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2RGB444 – JFIF YCbCr to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorJFIFYCC2RGB444(mllib_u8 *rgb, const mllib_u8 *y,
 const mllib_u8 *cb, const mllib_u8 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB444()` function performs color space conversion from YCbCr to RGB when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} R &= Y && + 1.40200 * (Cr - 128) \\ G &= Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128) \\ B &= Y + 1.77200 * (Cb - 128) \end{aligned}$$

**Parameters** The function takes the following arguments:

*rgb*     Pointer to destination RGB multi-component row. *rgb* must be 8-byte aligned.

*y*       Pointer to source Y component row. *y* must be 8-byte aligned.

*cb*      Pointer to source Cb component row. *cb* must be 8-byte aligned.

*cr*      Pointer to source Cr component row. *cr* must be 8-byte aligned.

*n*       Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCC2RGB444\_S16 – JFIF YCbCr to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorJFIFYCC2RGB444_S16(mllib_s16 *rgb,
 const mllib_s16 *y, const mllib_s16 *cb,
 const mllib_s16 *cr, mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCC2RGB444_S16()` function performs color space conversion from YCbCr to RGB when used in the JPEG File Interchange Format (JFIF).

Both the input YCbCr components and the output RGB components are supposed to be in the range of [0, 4095].

The following equation is used:

$$\begin{aligned} R &= Y && + 1.40200 * (Cr - 2048) \\ G &= Y - 0.34414 * (Cb - 2048) - 0.71414 * (Cr - 2048) \\ B &= Y + 1.77200 * (Cb - 2048) \end{aligned}$$

**Parameters** The function takes the following arguments:

- rgb*     Pointer to destination RGB multi-component row. *rgb* must be 8-byte aligned.
- y*       Pointer to source Y component row. *y* must be 8-byte aligned.
- cb*      Pointer to source Cb component row. *cb* must be 8-byte aligned.
- cr*      Pointer to source Cr component row. *cr* must be 8-byte aligned.
- n*       Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorJFIFYCC2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorJFIFYCC2RGB444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorJFIFYCCK2CMYK444 – JFIF YCbCr to CMYK color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorJFIFYCCK2CMYK444(mllib_u8 *cmyk,
 const mllib_u8 *y, const mllib_u8 *cb,
 const mllib_u8 *cr, const mllib_u8 *k,
 mllib_s32 n);
```

**Description** The `mllib_VideoColorJFIFYCCK2CMYK444()` function performs color space conversion from YCbCrK to CMYK when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} R &= Y && + 1.40200 * (Cr - 128) \\ G &= Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128) \\ B &= Y + 1.77200 * (Cb - 128) \\ C &= (255 - R) \\ M &= (255 - G) \\ Y &= (255 - B) \\ K &= K \end{aligned}$$

**Parameters** The function takes the following arguments:

- cmyk*     Pointer to destination CMYK multi-component row. *cmyk* must be 8-byte aligned.
- y*        Pointer to source Y component row. *y* must be 8-byte aligned.
- cb*        Pointer to source Cb component row. *cb* must be 8-byte aligned.
- cr*        Pointer to source Cr component row. *cr* must be 8-byte aligned.
- k*        Pointer to source K component row. *k* must be 8-byte aligned.
- n*        Length of Y, Cb, Cr, and K component rows. The length of the CMYK multi-component row must be 4\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorCMYK2JFIFYCCK444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorMerge2 – color conversion (color channel merge)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorMerge2(mllib_u8 *colors,
 const mllib_u8 *color1,const mllib_u8 *color2,
 mllib_s32 n);
```

**Description** The mllib\_VideoColorMerge2() function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1*      Pointer to first color component row. color1 must be 8-byte aligned.
- color2*      Pointer to second color component row. color2 must be 8-byte aligned.
- n*            Length of color1 and color2 arrays. Length of colors must be 2\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),[mllib\\_VideoColorMerge3\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#),[mllib\\_VideoColorMerge4\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#),[mllib\\_VideoColorSplit2\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#),[mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#),[mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#),[attributes\(5\)](#)

**Name** mllib\_VideoColorMerge2\_S16 – color conversion (color channel merge)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorMerge2_S16(mllib_s16 *colors,
 const mllib_s16 *color1, const mllib_s16 *color2,
 mllib_s32 n);
```

**Description** The mllib\_VideoColorMerge2\_S16() function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1*      Pointer to first color component row. color1 must be 8-byte aligned.
- color2*      Pointer to second color component row. color2 must be 8-byte aligned.
- n*            Length of color1 and color2 arrays. Length of colors must be 2\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorMerge3 – color conversion (color channel merge)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorMerge3(mllib_u8 *colors, const mllib_u8 *color1,
 const mllib_u8 *color2, const mllib_u8 *color3, mllib_s32 n);
```

**Description** The `mllib_VideoColorMerge3()` function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1*      Pointer to first color component row. color1 must be 8-byte aligned.
- color2*      Pointer to second color component row. color2 must be 8-byte aligned.
- color3*      Pointer to third color component row. color3 must be 8-byte aligned.
- n*            Length of color1, color2 and color3 arrays. Length of colors must be 3\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorMerge3\_S16 – color conversion (color channel merge)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorMerge3_S16(mllib_s16 *colors,
 const mllib_s16 *color1, const mllib_s16 *color2,
 const mllib_s16 *color3, mllib_s32 n);
```

**Description** The mllib\_VideoColorMerge3\_S16() function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors* Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1* Pointer to first color component row. color1 must be 8-byte aligned.
- color2* Pointer to second color component row. color2 must be 8-byte aligned.
- color3* Pointer to third color component row. color3 must be 8-byte aligned.
- n* Length of color1, color2 and color3 arrays. Length of colors must be 3\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#), [mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#), [mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorMerge4 – color conversion (color channel merge)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorMerge4(mllib_u8 *colors, const mllib_u8 *color1,
 const mllib_u8 *color2, const mllib_u8 *color3, const mllib_u8 *color4,
 mllib_s32 n);
```

**Description** The mllib\_VideoColorMerge4() function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1*      Pointer to first color component row. color1 must be 8-byte aligned.
- color2*      Pointer to second color component row. color2 must be 8-byte aligned.
- color3*      Pointer to third color component row. color3 must be 8-byte aligned.
- color4*      Pointer to fourth color component row. color4 must be 8-byte aligned.
- n*            Length of color1, color2, color3, and color4 arrays. Length of colors must be 4\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorMerge4\_S16 – color conversion (color channel merge)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorMerge4_S16(mllib_s16 *colors,
 const mllib_s16 *color1, const mllib_s16 *color2,
 const mllib_s16 *color3, const mllib_s16 *color4,
 mllib_s32 n);
```

**Description** The mllib\_VideoColorMerge4\_S16() function performs color channel merge.

**Parameters** The function takes the following arguments:

- colors* Pointer to colors multi-component row. colors must be 8-byte aligned.
- color1* Pointer to first color component row. color1 must be 8-byte aligned.
- color2* Pointer to second color component row. color2 must be 8-byte aligned.
- color3* Pointer to third color component row. color3 must be 8-byte aligned.
- color4* Pointer to fourth color component row. color4 must be 8-byte aligned.
- n* Length of color1, color2, color3, and color4 arrays. Length of colors must be 4\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#), [mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#), [mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorResizeABGR – image resize

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorResizeABGR(mllib_u32 *dst,
 const mllib_u32 *src, mllib_s32 dst_w,
 mllib_s32 dst_h, mllib_s32 dst_lb,
 mllib_s32 src_w, mllib_s32 src_h,
 mllib_s32 src_lb, mllib_filter filter);
```

**Description** The `mllib_VideoColorResizeABGR()` function resizes the source image with dimensions `src_w`, `src_h` into the destination image with dimensions `dst_w`, `dst_h` using nearest-neighbor, bilinear interpolation, or bicubic interpolation. The source buffer can contain multi-banded pixel stream, in which case, each band is resized independently. Edge conditions are handled according to the `MLIB_EDGE_SRC_EXTEND` scheme.

**Parameters** The function takes the following arguments:

- dst* Pointer to output image.
- src* Pointer to input image.
- dst\_w* Output image width in pixels.
- dst\_h* Output image height in rows.
- dst\_lb* Input image width in pixels.
- src\_w* Linebytes for input buffer.
- src\_h* Input image height in lines.
- src\_lb* Linebytes for input image.
- filter* Type of interpolation filter. It can be one of the following:
  - MLIB\_NEAREST
  - MLIB\_BILINEAR
  - MLIB\_BICUBIC

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VideoColorRGB2ABGR – color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorRGB2ABGR(mllib_u8 *abgr, const mllib_u8 *rgb,
 mllib_s32 n);
```

**Description** The `mllib_VideoColorRGB2ABGR()` function performs RGB to ABGR color order conversion.

**Parameters** The function takes the following arguments:

- abgr*      Pointer to ABGR row.
- rgb*       Pointer to RGB row.
- n*         Number of pixels.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2RGB\(3MLIB\)](#), [mllib\\_VideoColorARGB2RGB\(3MLIB\)](#), [mllib\\_VideoColorRGB2ARGB\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VideoColorRGB2ARGB – color conversion

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_VideoColorRGB2ARGB(mlib_u8 *argb, const mlib_u8 *rgb,
 mlib_s32 n);
```

**Description** The `mlib_VideoColorRGB2ARGB()` function performs RGB to ARGB color order conversion.

**Parameters** The function takes the following arguments:

*argb*      Pointer to ARGB row.

*rgb*        Pointer to RGB row.

*n*          Number of pixels.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoColorABGR2RGB\(3MLIB\)](#), [mlib\\_VideoColorARGB2RGB\(3MLIB\)](#),  
[mlib\\_VideoColorRGB2ABGR\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorRGB2JFIFYCC420 – RGB to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorRGB2JFIFYCC420(mllib_u8 *y0, mllib_u8 *y1,
 mllib_u8 *cb, mllib_u8 *cr, const mllib_u8 *rgb0, const mllib_u8 *rgb1,
 mllib_s32 n);
```

**Description** The `mllib_VideoColorRGB2JFIFYCC420()` function performs color space conversion from RGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- y0* Pointer to upper destination Y component row. *y0* must be 8-byte aligned.
- y1* Pointer to lower destination Y component row. *y1* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- rgb0* Pointer to upper source RGB multi-component row. *rgb0* must be 8-byte aligned.
- rgb1* Pointer to lower source RGB multi-component row. *rgb1* must be 8-byte aligned.
- n* Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorABGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC422\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorRGB2JFIFYCC422 – RGB to JFIF YCbCr color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorRGB2JFIFYCC422(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *rgb, mllib_s32 n);
```

**Description** The `mllib_VideoColorRGB2JFIFYCC422()` function performs color space conversion from RGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*y*        Pointer to destination Y component row. *y* must be 8-byte aligned.

*cb*       Pointer to destination Cb component row. *cb* must be 8-byte aligned.

*cr*       Pointer to destination Cr component row. *cr* must be 8-byte aligned.

*rgb*      Pointer to source RGB multi-component row. *rgb* must be 8-byte aligned.

*n*        Length of Y component row. *n* must be even. The length of Cb and Cr component rows must be *n*/2. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorABGR2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC420\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC422\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC420\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorRGB2JFIFYCC444 – RGB to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorRGB2JFIFYCC444(mllib_u8 *y, mllib_u8 *cb,
 mllib_u8 *cr, const mllib_u8 *rgb, mllib_s32 n);
```

**Description** The `mllib_VideoColorRGB2JFIFYCC444()` function performs color space conversion from RGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128 \end{aligned}$$

**Parameters** The function takes the following arguments:

- y* Pointer to destination Y component row. *y* must be 8-byte aligned.
- cb* Pointer to destination Cb component row. *cb* must be 8-byte aligned.
- cr* Pointer to destination Cr component row. *cr* must be 8-byte aligned.
- rgb* Pointer to source RGB multi-component row. *rgb* must be 8-byte aligned.
- n* Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorRGB2JFIFYCC444\_S16 – RGB to JFIF YCbCr color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorRGB2JFIFYCC444_S16(mllib_s16 *y, mllib_s16 *cb,
 mllib_s16 *cr, const mllib_s16 *rgb, mllib_s32 n);
```

**Description** The `mllib_VideoColorRGB2JFIFYCC444_S16()` function performs color space conversion from RGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

Both the input RGB components and the output YCbCr components are supposed to be in the range of [0, 4095].

The following equation is used:

$$\begin{aligned} Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\ Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 2048 \\ Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 2048 \end{aligned}$$

**Parameters** The function takes the following arguments:

*y*        Pointer to destination Y component row. *y* must be 8-byte aligned.  
*cb*       Pointer to destination Cb component row. *cb* must be 8-byte aligned.  
*cr*       Pointer to destination Cr component row. *cr* must be 8-byte aligned.  
*rgb*      Pointer to source RGB multi-component row. *rgb* must be 8-byte aligned.  
*n*        Length of Y component row. The length of Cb and Cr component rows must be *n*. The length of the RGB multi-component row must be 3\**n*.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGR2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorARGB2JFIFYCC444\(3MLIB\)](#), [mllib\\_VideoColorRGB2JFIFYCC444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorRGBARint\_to\_ABGRint – convert RGBA interleaved to ABGR

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorRGBARint_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u32 *RGBA,mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The RGBA pixel stream is broken apart and recombined into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions *w* and *h*. Within each 32-bit input word, the component ordering is R (bits 31-24), G (bits 23-16), B (bits 15-8), and A (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

**Parameters** The function takes the following arguments:

- ABGR*     Pointer to output buffer.
- RGBA*     Pointer to input buffer.
- w*         Image width in pixels.
- h*         Image height in lines.
- dlb*       Linebytes for output buffer.
- slb*       Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorABGRint\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorBGRARint\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorRGBInt\_to\_ABGRInt – convert RGB interleaved to ABGR interleaved

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_VideoColorRGBInt_to_ABGRInt(mllib_u32 *ABGR,
 const mllib_u8 *RGB, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The interleaved RGB stream, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 24-bit input pixel, the component ordering is R (bits 23-16), G (bits 15-8), and B (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>RGB</i>     | Pointer to input buffer.     |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorRGBseq_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorBGRInt_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorRGBXint_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorRGBXint_to_ARGBInt(3MLIB)`,  
`mllib_VideoColorXRGBInt_to_ABGRInt(3MLIB)`,  
`mllib_VideoColorXRGBInt_to_ARGBInt(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoColorRGBInt\_to\_BGRAInt – convert RGB interleaved to BGRA interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorRGBInt_to_BGRAInt(mllib_u8 *bgra,
 const mllib_u8 *rgb, const mllib_u8 *a_array,
 mllib_u8 a_const, mllib_s32 w, mllib_s32 h,
 mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** The interleaved RGB stream and the A values are combined into an interleaved BGRA byte stream.

The alpha values for this function work in the following fashion:

- If the `a_array` pointer is not NULL, the values are taken from there. It has to have the at least 1/3 the dimension of the RGB buffer.
- If the `a_array` pointer is NULL, the alpha values for every pixel are set to `a_const`.

In other words, this function's inner loop works like this:

```
bgra[0] = rgb[2];
bgra[1] = rgb[1];
bgra[2] = rgb[0];
bgra[3] = (a_array == NULL) ? a_const : a_array[0];
```

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>bgra</i>    | Pointer to the output BGRA buffer. |
| <i>rgb</i>     | Pointer to the input RGB buffer.   |
| <i>a_array</i> | Pointer to the alpha buffer.       |
| <i>a_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes of the output buffer.    |
| <i>slb</i>     | Linebytes of the input buffer.     |
| <i>alb</i>     | Linebytes of the alpha buffer.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorRGBInt\\_to\\_ABGRInt\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorRGBseq\_to\_ABGRint – convert RGB sequential to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorRGBseq_to_ABGRint(mllib_u32 *ABGR, const mllib_u8 *R,
 const mllib_u8 *G, const mllib_u8 *B,
 const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h,
 mllib_s32 dlb, mllib_s32 slb);
```

**Description** The R, G, and B streams, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>R</i>       | Pointer to input R buffer.   |
| <i>G</i>       | Pointer to input G buffer.   |
| <i>B</i>       | Pointer to input B buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorRGBint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorBGRint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorRGBXint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorRGBXint_to_ARGBint(3MLIB)`,  
`mllib_VideoColorXRGBint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorXRGBint_to_ARGBint(3MLIB)`, `attributes(5)`

**Name** mlib\_VideoColorRGBXint\_to\_ABGRint – convert RGBX interleaved to ABGR interleaved

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
void mlib_VideoColorRGBXint_to_ABGRint(mlib_u32 *ABGR,
 const mlib_u32 *RGBX, const mlib_u8 *A_array,
 mlib_u8 A_const, mlib_s32 w,
 mlib_s32 h, mlib_s32 dlb,
 mlib_s32 slb, mlib_s32 alb);
```

**Description** The interleaved RGBX stream and the alpha values are combined into an interleaved A, B, G, R output stream. Within each 32-bit input pixel, the component ordering is R (bits 31-24), G (bits 23-16), and B (bits 15-8). Within each 32-bit output pixel, the component bordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0). The alpha values for this function work in the following fashion: if *A\_array* is not NULL, the values are taken from the corresponding locations in the alpha array, otherwise a constant alpha value, specified by *A\_const*, is store in each output pixel. Each element in the alpha array is an unsigned byte. *w* and *h* define the dimensions of the region of the buffers to be processed. The linebyte parameters are used to advance the data pointers for each of the buffers.

**Parameters** The function takes the following arguments:

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>ABGR</i>    | Pointer to output buffer (word-aligned).         |
| <i>RGBX</i>    | Pointer to input buffer (word-aligned).          |
| <i>A_array</i> | Pointer to array of alpha values (byte-aligned). |
| <i>A_const</i> | Constant alpha value (range = 0..255).           |
| <i>w</i>       | Image width in pixels.                           |
| <i>h</i>       | Image height in lines.                           |
| <i>dlb</i>     | Linebytes for output buffer.                     |
| <i>slb</i>     | Linebytes for input buffer.                      |
| <i>alb</i>     | Linebytes for alpha buffer.                      |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorRGBseq_to_ABGRint(3MLIB)`,  
`mllib_VideoColorRGBint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorBGRint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorRGBXint_to_ARGBint(3MLIB)`,  
`mllib_VideoColorXRGBint_to_ABGRint(3MLIB)`,  
`mllib_VideoColorXRGBint_to_ARGBint(3MLIB)`, `attributes(5)`

**Name** mlib\_VideoColorRGBXint\_to\_ARGBint – convert RGBX interleaved to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
void mlib_VideoColorRGBXint_to_ARGBint(mlib_u32 *ARGB,
 const mlib_u32 *RGBX, const mlib_u8 *A_array,
 mlib_u8 A_const, mlib_s32 w,
 mlib_s32 h, mlib_s32 dlb,
 mlib_s32 slb, mlib_s32 alb);
```

**Description** Similar to `mlib_VideoColorRGBXint_to_ABGRint()` except that the output component ordering is: A (bits 31-24), R (bits 23-16), G (bits 15-8), and B (bits 7-0).

**Parameters** The function takes the following arguments:

*ARGB*      Pointer to output buffer (word-aligned).  
*RGBX*      Pointer to input buffer (word-aligned).  
*A\_array*    Pointer to array of alpha values (byte-aligned).  
*A\_const*    Constant alpha value (range = 0..255).  
*w*          Image width in pixels.  
*h*          Image height in lines.  
*dlb*        Linebytes for output buffer.  
*slb*        Linebytes for input buffer.  
*alb*        Linebytes for alpha buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoColorRGBseq\\_to\\_ABGRint\(3MLIB\)](#),  
[mlib\\_VideoColorRGBint\\_to\\_ABGRint\(3MLIB\)](#),  
[mlib\\_VideoColorBGRint\\_to\\_ABGRint\(3MLIB\)](#),  
[mlib\\_VideoColorRGBXint\\_to\\_ABGRint\(3MLIB\)](#),  
[mlib\\_VideoColorXRGBint\\_to\\_ABGRint\(3MLIB\)](#),  
[mlib\\_VideoColorXRGBint\\_to\\_ARGBint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorSplit2 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit2(mllib_u8 *color1, mllib_u8 *color2,
 const mllib_u8 *colors, mllib_s32 n);
```

**Description** The mllib\_VideoColorSplit2() function performs color channel split.

The elements of the colors array are alternately copied into the color1 array and color2 array.

**Parameters** The function takes the following arguments:

*color1*      Pointer to first color component row. color1 must be 8-byte aligned.

*color2*      Pointer to second color component row. color2 must be 8-byte aligned.

*colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.

*n*            Length of color1 and color2 arrays. Length of colors must be 2\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorSplit2\_S16 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit2_S16(mllib_s16 *color1, mllib_s16 *color2,
 const mllib_s16 *colors, mllib_s32 n);
```

**Description** The `mllib_VideoColorSplit2_S16()` function performs color channel split.  
The elements of the colors array are alternately copied into the color1 array and color2 array.

**Parameters** The function takes the following arguments:

*color1*     Pointer to first color component row. color1 must be 8-byte aligned.

*color2*     Pointer to second color component row. color2 must be 8-byte aligned.

*colors*     Pointer to colors multi-component row. colors must be 8-byte aligned.

*n*           Length of color1 and color2 arrays. Length of colors must be 2\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorSplit3 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit3(mllib_u8 *color1, mllib_u8 *color2,
 mllib_u8 *color3, const mllib_u8 *colors, mllib_s32 n);
```

**Description** The mllib\_VideoColorSplit3() function performs color channel split.

The elements of the colors array are selected in consecutive groups of three. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and the third element, in the color3 array. This process is repeated untill the end of the colors array is reached.

**Parameters** The function takes the following arguments:

- color1*     Pointer to first color component row. color1 must be 8-byte aligned.
- color2*     Pointer to second color component row. color2 must be 8-byte aligned.
- colors*     Pointer to colors multi-component row. colors must be 8-byte aligned.
- n*           Length of color1, color2, and color3 arrays. Length of colors must be 3\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#), [mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorSplit3\_S16 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit3_S16(mllib_s16 *color1, mllib_s16 *color2,
 mllib_s16 *color3, const mllib_s16 *colors, mllib_s32 n);
```

**Description** The `mllib_VideoColorSplit3_S16()` function performs color channel split.

The elements of the colors array are selected in consecutive groups of three. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and the third element, in the color3 array. This process is repeated until the end of the colors array is reached.

**Parameters** The function takes the following arguments:

*color1*     Pointer to first color component row. color1 must be 8-byte aligned.

*color2*     Pointer to second color component row. color2 must be 8-byte aligned.

*colors*     Pointer to colors multi-component row. colors must be 8-byte aligned.

*n*           Length of color1, color2, and color3 arrays. Length of colors must be 3\*n.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#), [mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorSplit4 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit4(mllib_u8 *color1, mllib_u8 *color2,
 mllib_u8 *color3, mllib_u8 *color4, const mllib_u8 *colors, mllib_s32 n);
```

**Description** The mllib\_VideoColorSplit4() function performs color channel split.

The elements of the colors array are selected in consecutive groups of four. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and so on. This process is repeated untill the end of the colors array is reached.

**Parameters** The function takes the following arguments:

- color1*     Pointer to first color component row. color1 must be 8-byte aligned.
- color2*     Pointer to second color component row. color2 must be 8-byte aligned.
- color3*     Pointer to third color component row. color3 must be 8-byte aligned.
- color4*     Pointer to fourth color component row. color4 must be 8-byte aligned.
- colors*     Pointer to colors multi-component row. colors must be 8-byte aligned.
- n*           Length of color1, color2, color3, and color4 arrays. Length of colors must be 4\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorSplit3\(3MLIB\)](#), [mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#),  
[mllib\\_VideoColorSplit4\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorSplit4\_S16 – color conversion (color channel split)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorSplit4_S16(mllib_s16 *color1,
 mllib_s16 *color2,mllib_s16 *color3,
 mllib_s16 *color4, const mllib_s16 *colors,
 mllib_s32 n);
```

**Description** The mllib\_VideoColorSplit4\_S16() function performs color channel split.

The elements of the colors array are selected in consecutive groups of four. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and so on. This process is repeated untill the end of the colors array is reached.

**Parameters** The function takes the following arguments:

*color1*      Pointer to first color component row. color1 must be 8-byte aligned.

*color2*      Pointer to second color component row. color2 must be 8-byte aligned.

*colors*      Pointer to colors multi-component row. colors must be 8-byte aligned.

*n*            Length of color1, color2, color3, and color4 arrays. Length of colors must be 4\*n.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorMerge2\(3MLIB\)](#), [mllib\\_VideoColorMerge2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge3\(3MLIB\)](#), [mllib\\_VideoColorMerge3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorMerge4\(3MLIB\)](#), [mllib\\_VideoColorMerge4\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit2\(3MLIB\)](#), [mllib\\_VideoColorSplit2\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit3\(3MLIB\)](#), [mllib\\_VideoColorSplit3\\_S16\(3MLIB\)](#), [mllib\\_VideoColorSplit4\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorUYV444int\_to\_ABGRint – color convert UYV interleaved to ABGR interleaved

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_VideoColorUYV444int_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *UYV, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The UYV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the *Y* buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>UYV</i>     | Pointer to input buffer.     |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),``attributes(5)`

**Name** mllib\_VideoColorUYV444int\_to\_ARGBint – color convert UYV interleaved to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorUYV444int_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u8 *UYV, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The UYV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the *Y* buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.    |
| <i>UYV</i>     | Pointer to input buffer.     |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorUYV444int\_to\_UYVY422int – convert UYV interleaved with subsampling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorUYV444int_to_UYVY422int(mllib_u32 *UYVY,
 const mllib_u8 *UYV, mllib_s32 w,mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The UYV pixel stream is broken apart and recombined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The UYV buffer has dimensions *w* and *h*. Dimension *w* is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is U[r][c], Y[r][c], V[r][c], U[r][c+1], Y[r][c+1], V[r][c+1], ...

The following equation is used:

$$\begin{aligned} \text{UYVY}[r][c/2] = & ((\text{U}[r][c] + \text{U}[r][c+1]) / 2) \ll 24) \mid \\ & (\text{Y}[r][c] \ll 16) \mid \\ & (((\text{V}[r][c] + \text{V}[r][c+1]) / 2) \ll 8) \mid \\ & (\text{Y}[r][c+1]) \end{aligned}$$

where *r* = 0, 1, 2, ..., *h*-1; and *c* = 0, 2, 4, ..., *w*-2.

**Parameters** The function takes the following arguments:

- UYVY*     Pointer to output buffer.
- UYV*     Pointer to input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*     Linebytes for output buffer.
- slb*     Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV444seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_UYVY422int\(3MLIB\)](#),



```
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
mlib_VideoColorUYV444int_to_YUYV422int(3MLIB), attributes(5)
```

**Name** mllib\_VideoColorUYV444int\_to\_YUYV422int – convert UYV interleaved with subsampling

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorUYV444int_to_YUYV422int(mllib_u32 *YUYV,
 const mllib_u8 *UYV,mllib_s32 w,mllib_s32 h,
 mllib_s32 dlb, mllib_s32 slb);
```

**Description** The UYV pixel stream is broken apart and recombined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The UYV buffer has dimensions *w* and *h*. Dimension *w* is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is U[r][c], Y[r][c], V[r][c], U[r][c+1], Y[r][c+1], V[r][c+1], ...

The following equation is used:

$$\begin{aligned} \text{YUYV}[r][c/2] = & (Y[r][c] \ll 24) \mid \\ & (((U[r][c] + U[r][c+1]) / 2) \ll 16) \mid \\ & (Y[r][c+1] \ll 8) \mid \\ & (((V[r][c] + V[r][c+1]) / 2)) \end{aligned}$$

where *r* = 0, 1, 2, ..., *h*-1; and *c* = 0, 2, 4, ..., *w*-2.

**Parameters** The function takes the following arguments:

- YUYV*     Pointer to output buffer.
- UYV*     Pointer to input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*     Linebytes for output buffer.
- slb*     Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV444seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_UYVY422int\(3MLIB\)](#),

```
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
mlib_VideoColorUYV444int_to_UYVY422int(3MLIB), attributes(5)
```

**Name** mllib\_VideoColorUYVY422int\_to\_ABGRint – color convert UYVY interleaved to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorUYVY422int_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u32 *UYVY, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The UYVY pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The UYVY buffer has dimensions  $w/2$  and  $h$ . The ABGR buffer has dimensions  $w$  and  $h$ . Dimension  $w$  is assumed to be even.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>UYVY</i>    | Pointer to input buffer.     |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY444int_to_ABGRint(3MLIB),``attributes(5)`

**Name** mllib\_VideoColorUYVY422int\_to\_ARGBint – color convert UYVY interleaved to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorUYVY422int_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u32 *UYVY, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h,
 mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** The UYVY pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The UYVY buffer has dimensions  $w/2$  and  $h$ . The ARGB buffer has dimensions  $w$  and  $h$ . Dimension  $w$  is assumed to be even.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.    |
| <i>UYVY</i>    | Pointer to input buffer.     |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),``attributes(5)`

**Name** mllib\_VideoColorXRGBint\_to\_ABGRint – convert XRGB interleaved to ABGR interleaved

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorXRGBint_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u32 *XRGB, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h,
 mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** Similar to mllib\_VideoColorRGBXint\_to\_ABGRint except that the input component ordering is: R (bits 23-16), G (bits 15-8), and B (bits 7-0).

**Parameters** The function takes the following arguments:

- ABGR* Pointer to output buffer (word-aligned).
- XRGB* Pointer to input buffer (word-aligned).
- A\_array* Pointer to array of alpha values (byte-aligned).
- A\_const* Constant alpha value (range = 0..255).
- w* Image width in pixels.
- h* Image height in lines.
- dlb* Linebytes for output buffer.
- slb* Linebytes for input buffer.
- alb* Linebytes for alphabuffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorRGBseq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorRGBint\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorBGRint\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorRGBXint\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorRGBXint\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorXRGBint\\_to\\_ARGBint\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_VideoColorXRGBInt\_to\_ARGBInt – convert XRGB interleaved to ARGB interleaved

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
void mlib_VideoColorXRGBInt_to_ARGBInt(mlib_u32 *ARGB,
 const mlib_u32 *XRGB, const mlib_u8 *A_array,
 mlib_u8 A_const, mlib_s32 w, mlib_s32 h,
 mlib_s32 dlb, mlib_s32 slb, mlib_s32 alb);
```

**Description** Similar to mlib\_VideoColorRGBXint\_to\_ARGBInt except that the input component ordering is: R (bits 23-16), G (bits 15-8), and B (bits 7-0).

**Parameters** The function takes the following arguments:

*ARGB* Pointer to output buffer (word-aligned).  
*XRGB* Pointer to input buffer (word-aligned).  
*A\_array* Pointer to array of alpha values (byte-aligned).  
*A\_const* Constant alpha value (range = 0..255).  
*w* Image width in pixels.  
*h* Image height in lines.  
*dlb* Linebytes for output buffer.  
*slb* Linebytes for input buffer.  
*alb* Linebytes for alphabuffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoColorRGBseq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mlib\\_VideoColorRGBInt\\_to\\_ABGRInt\(3MLIB\)](#),  
[mlib\\_VideoColorBGRInt\\_to\\_ABGRInt\(3MLIB\)](#),  
[mlib\\_VideoColorRGBXint\\_to\\_ABGRInt\(3MLIB\)](#),  
[mlib\\_VideoColorRGBXint\\_to\\_ARGBInt\(3MLIB\)](#),  
[mlib\\_VideoColorXRGBInt\\_to\\_ABGRInt\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2ABGR411 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR411(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR411()` function performs YUV to RGB color conversion used in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),

```
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
```

**Name** mllib\_VideoColorYUV2ABGR420 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR420()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),

```
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
```

**Name** mllib\_VideoColorYUV2ABGR420\_W – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420_W(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 abgr_stride, mllib_s32 y_stride, mllib_s32 uv_stride,
 mllib_s32 left, mllib_s32 top, mllib_s32 right,
 mllib_s32 bottom);
```

**Description** The `mllib_VideoColorYUV2ABGR420_W()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                    |                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>abgr</i>        | Pointer to the destination packed ABGR image. <i>abgr</i> must be 8-byte aligned.                                  |
| <i>y</i>           | Pointer to the source Y component. <i>y</i> must be 8-byte aligned.                                                |
| <i>u</i>           | Pointer to the source U component. <i>u</i> must be 4-byte aligned.                                                |
| <i>v</i>           | Pointer to the source V component. <i>v</i> must be 4-byte aligned.                                                |
| <i>width</i>       | Width of the image. <i>width</i> must be a multiple of 8.                                                          |
| <i>height</i>      | Height of the image. <i>height</i> must be a multiple of 2.                                                        |
| <i>abgr_stride</i> | Stride, in bytes, between adjacent rows in the ABGR image. <i>abgr_stride</i> must be a multiple of 8.             |
| <i>y_stride</i>    | Stride, in bytes, between adjacent rows in the Y component image. <i>y_stride</i> must be a multiple of 8.         |
| <i>uv_stride</i>   | Stride, in bytes, between adjacent rows in the U and V component images. <i>uv_stride</i> must be a multiple of 8. |
| <i>left</i>        | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>top</i>         | Top border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                            |
| <i>right</i>       | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>bottom</i>      | Bottom border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                         |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR420\\_WX2\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_WX3\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X2\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X3\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2ABGR420\_WX2 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420_WX2(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 abgr_stride, mllib_s32 y_stride, mllib_s32 uv_stride,
 mllib_s32 left, mllib_s32 top, mllib_s32 right,
 mllib_s32 bottom);
```

**Description** The `mllib_VideoColorYUV2ABGR420_WX2()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping and 2X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                    |                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>abgr</i>        | Pointer to the destination packed ABGR image. <i>abgr</i> must be 8-byte aligned.                                  |
| <i>y</i>           | Pointer to the source Y component. <i>y</i> must be 8-byte aligned.                                                |
| <i>u</i>           | Pointer to the source U component. <i>u</i> must be 4-byte aligned.                                                |
| <i>v</i>           | Pointer to the source V component. <i>v</i> must be 4-byte aligned.                                                |
| <i>width</i>       | Width of the image. <i>width</i> must be a multiple of 8.                                                          |
| <i>height</i>      | Height of the image. <i>height</i> must be a multiple of 2.                                                        |
| <i>abgr_stride</i> | Stride, in bytes, between adjacent rows in the ABGR image. <i>abgr_stride</i> must be a multiple of 8.             |
| <i>y_stride</i>    | Stride, in bytes, between adjacent rows in the Y component image. <i>y_stride</i> must be a multiple of 8.         |
| <i>uv_stride</i>   | Stride, in bytes, between adjacent rows in the U and V component images. <i>uv_stride</i> must be a multiple of 8. |
| <i>left</i>        | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>top</i>         | Top border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                            |
| <i>right</i>       | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>bottom</i>      | Bottom border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                         |



**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR420\\_W\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_WX3\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X2\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X3\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2ABGR420\_WX3 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420_WX3(mllib_u8 *abgr,
 const mllib_u8 *y, const mllib_u8 *u, const mllib_u8 *v,
 mllib_s32 width, mllib_s32 height,
 mllib_s32 abgr_stride, mllib_s32 y_stride,
 mllib_s32 uv_stride, mllib_s32 left,
 mllib_s32 top, mllib_s32 right, mllib_s32 bottom);
```

**Description** The `mllib_VideoColorYUV2ABGR420_WX3()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping and 3X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                    |                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>abgr</i>        | Pointer to the destination packed ABGR image. <i>abgr</i> must be 8-byte aligned.                                  |
| <i>y</i>           | Pointer to the source Y component. <i>y</i> must be 8-byte aligned.                                                |
| <i>u</i>           | Pointer to the source U component. <i>u</i> must be 4-byte aligned.                                                |
| <i>v</i>           | Pointer to the source V component. <i>v</i> must be 4-byte aligned.                                                |
| <i>width</i>       | Width of the image. <i>width</i> must be a multiple of 8.                                                          |
| <i>height</i>      | Height of the image. <i>height</i> must be a multiple of 2.                                                        |
| <i>abgr_stride</i> | Stride, in bytes, between adjacent rows in the ABGR image. <i>abgr_stride</i> must be a multiple of 8.             |
| <i>y_stride</i>    | Stride, in bytes, between adjacent rows in the Y component image. <i>y_stride</i> must be a multiple of 8.         |
| <i>uv_stride</i>   | Stride, in bytes, between adjacent rows in the U and V component images. <i>uv_stride</i> must be a multiple of 8. |
| <i>left</i>        | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>top</i>         | Top border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                            |
| <i>right</i>       | Left border of clipping window. $0 \leq \text{left} < \text{right} \leq \text{width}$ .                            |
| <i>bottom</i>      | Bottom border of clipping window. $0 \leq \text{top} < \text{bottom} \leq \text{height}$ .                         |

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR420\\_W\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_WX2\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X2\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\\_X3\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2ABGR420\_X2 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420_X2(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 abgr_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR420_X2()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with 2X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image. *abgr* must be 8-byte aligned.
- y* Pointer to the source Y component. *y* must be 8-byte aligned.
- u* Pointer to the source U component. *u* must be 4-byte aligned.
- v* Pointer to the source V component. *v* must be 4-byte aligned.
- width* Width of the image. *width* must be a multiple of 8.
- height* Height of the image. *height* must be a multiple of 2.
- abgr\_stride* Stride, in bytes, between adjacent rows in the ABGR image. *abgr\_stride* must be a multiple of 8.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image. *y\_stride* must be a multiple of 8.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images. *uv\_stride* must be a multiple of 8.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoColorYUV2ABGR420_W(3MLIB)`, `mlib_VideoColorYUV2ABGR420_WX2(3MLIB)`,  
`mlib_VideoColorYUV2ABGR420_WX3(3MLIB)`, `mlib_VideoColorYUV2ABGR420_X3(3MLIB)`,  
`attributes(5)`

**Name** mllib\_VideoColorYUV2ABGR420\_X3 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR420_X3(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 abgr_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR420_X3()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with 3X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image. *abgr* must be 8-byte aligned.
- y* Pointer to the source Y component. *y* must be 8-byte aligned.
- u* Pointer to the source U component. *u* must be 4-byte aligned.
- v* Pointer to the source V component. *v* must be 4-byte aligned.
- width* Width of the image. *width* must be a multiple of 8.
- height* Height of the image. *height* must be a multiple of 2.
- abgr\_stride* Stride, in bytes, between adjacent rows in the ABGR image. *abgr\_stride* must be a multiple of 8.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image. *y\_stride* must be a multiple of 8.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images. *uv\_stride* must be a multiple of 8.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoColorYUV2ABGR420_W(3MLIB)`, `mlib_VideoColorYUV2ABGR420_WX2(3MLIB)`,  
`mlib_VideoColorYUV2ABGR420_WX3(3MLIB)`, `mlib_VideoColorYUV2ABGR420_X2(3MLIB)`,  
`attributes(5)`

**Name** mllib\_VideoColorYUV2ABGR422 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR422(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR422()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),



```
mlib_VideoColorYUV2RGB420(3MLIB),mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2ABGR444 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2ABGR444(mllib_u8 *abgr, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 yuv_stride);
```

**Description** The `mllib_VideoColorYUV2ABGR444()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

**Parameters** The function takes the following arguments:

- abgr* Pointer to the destination packed ABGR image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- yuv\_stride* Stride, in bytes, between adjacent rows in the source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2ARGB411 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorYUV2ARGB411(mllib_u8 *argb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ARGB411()` function performs YUV to RGB color conversion used in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| <i>argb</i>       | Pointer to the destination packed ARGB image.                            |
| <i>y</i>          | Pointer to the source Y component.                                       |
| <i>u</i>          | Pointer to the source U component.                                       |
| <i>v</i>          | Pointer to the source V component.                                       |
| <i>width</i>      | Width of the image.                                                      |
| <i>height</i>     | Height of the image.                                                     |
| <i>rgb_stride</i> | Stride, in bytes, between adjacent rows in the destination image.        |
| <i>y_stride</i>   | Stride, in bytes, between adjacent rows in the Y component image.        |
| <i>uv_stride</i>  | Stride, in bytes, between adjacent rows in the U and V component images. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),

```
mllib_VideoColorYUV2RGB420(3MLIB),mllib_VideoColorYUV2RGB422(3MLIB),
mllib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2ARGB420 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorYUV2ARGB420(mllib_u8 *argb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ARGB420()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| <i>argb</i>       | Pointer to the destination packed ARGB image.                            |
| <i>y</i>          | Pointer to the source Y component.                                       |
| <i>u</i>          | Pointer to the source U component.                                       |
| <i>v</i>          | Pointer to the source V component.                                       |
| <i>width</i>      | Width of the image.                                                      |
| <i>height</i>     | Height of the image.                                                     |
| <i>rgb_stride</i> | Stride, in bytes, between adjacent rows in the destination image.        |
| <i>y_stride</i>   | Stride, in bytes, between adjacent rows in the Y component image.        |
| <i>uv_stride</i>  | Stride, in bytes, between adjacent rows in the U and V component images. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),

```
mllib_VideoColorYUV2RGB420(3MLIB),mllib_VideoColorYUV2RGB422(3MLIB),
mllib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2ARGB422 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorYUV2ARGB422(mllib_u8 *argb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2ARGB422()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| <i>argb</i>       | Pointer to the destination packed ARGB image.                            |
| <i>y</i>          | Pointer to the source Y component.                                       |
| <i>u</i>          | Pointer to the source U component.                                       |
| <i>v</i>          | Pointer to the source V component.                                       |
| <i>width</i>      | Width of the image.                                                      |
| <i>height</i>     | Height of the image.                                                     |
| <i>rgb_stride</i> | Stride, in bytes, between adjacent rows in the destination image.        |
| <i>y_stride</i>   | Stride, in bytes, between adjacent rows in the Y component image.        |
| <i>uv_stride</i>  | Stride, in bytes, between adjacent rows in the U and V component images. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#),

```
mllib_VideoColorYUV2RGB420(3MLIB),mllib_VideoColorYUV2RGB422(3MLIB),
mllib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```



**Name** mllib\_VideoColorYUV2ARGB444 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoColorYUV2ARGB444(mllib_u8 *argb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 yuv_stride);
```

**Description** The `mllib_VideoColorYUV2ARGB444()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

**Parameters** The function takes the following arguments:

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>argb</i>       | Pointer to the destination packed ARGB image.                     |
| <i>y</i>          | Pointer to the source Y component.                                |
| <i>u</i>          | Pointer to the source U component.                                |
| <i>v</i>          | Pointer to the source V component.                                |
| <i>width</i>      | Width of the image.                                               |
| <i>height</i>     | Height of the image.                                              |
| <i>rgb_stride</i> | Stride, in bytes, between adjacent rows in the destination image. |
| <i>yuv_stride</i> | Stride, in bytes, between adjacent rows in the source image.      |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB444\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV2RGB411 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2RGB411(mllib_u8 *rgb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2RGB411()` function performs YUV to RGB color conversion used in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

- rgb* Pointer to the destination RGB image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#),

```
mlib_VideoColorYUV2RGB420(3MLIB),mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2RGB420 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorYUV2RGB420(mllib_u8 *rgb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2RGB420()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

**Parameters** The function takes the following arguments:

- rgb* Pointer to the destination RGB image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#),

```
mlib_VideoColorYUV2RGB411(3MLIB),mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2RGB422 – YUV to RGB color conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoColorYUV2RGB422(mllib_u8 *rgb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 y_stride, mllib_s32 uv_stride);
```

**Description** The `mllib_VideoColorYUV2RGB422()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters** The function takes the following arguments:

- rgb* Pointer to the destination RGB image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- y\_stride* Stride, in bytes, between adjacent rows in the Y component image.
- uv\_stride* Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#),

```
mlib_VideoColorYUV2RGB411(3MLIB),mlib_VideoColorYUV2RGB420(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV2RGB444 – YUV to RGB color conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoColorYUV2RGB444(mllib_u8 *rgb, const mllib_u8 *y,
 const mllib_u8 *u, const mllib_u8 *v, mllib_s32 width, mllib_s32 height,
 mllib_s32 rgb_stride, mllib_s32 yuv_stride);
```

**Description** The `mllib_VideoColorYUV2RGB444()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

**Parameters** The function takes the following arguments:

- rgb* Pointer to the destination RGB image.
- y* Pointer to the source Y component.
- u* Pointer to the source U component.
- v* Pointer to the source V component.
- width* Width of the image.
- height* Height of the image.
- rgb\_stride* Stride, in bytes, between adjacent rows in the destination image.
- yuv\_stride* Stride, in bytes, between adjacent rows in the source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV2ABGR411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ABGR444\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB422\(3MLIB\)](#), [mllib\\_VideoColorYUV2ARGB444\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB411\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB420\(3MLIB\)](#), [mllib\\_VideoColorYUV2RGB422\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorYUV411seq\_to\_ABGRint – color convert YUV sequential to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV411seq_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 yalb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/4$  and  $h$ . Dimension  $w$  is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |
| <i>uvlb</i>    | Linebytes for U and V buffers.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV411seq\_to\_ARGBint – color convert YUV sequential to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV411seq_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u8 *Y, const mllib_u8 *U,
 const mllib_u8 *V, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb,
 mllib_s32 yalb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/4$  and  $h$ . Dimension  $w$  is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |
| <i>uvlb</i>    | Linebytes for U and V buffers.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRInt\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV411seq\_to\_UYVY422int – convert YUV sequential to interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV411seq_to_UYVY422int(mllib_u32 *UYVY,
const mllib_u8 *Y, const mllib_u8 *U,
const mllib_u8 *V, mllib_s32 w,
mllib_s32 h, mllib_s32 dlb,
mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/4$  and  $h$ . Dimension  $w$  is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The following equation is used:

$$\begin{aligned} \text{UYVY}[r][c/2] &= (\text{U}[r][c/4] \ll 24) \mid \\ &\quad (\text{Y}[r][c] \ll 16) \mid \\ &\quad (\text{V}[r][c/4] \ll 8) \mid \\ &\quad (\text{Y}[r][c+1]) \end{aligned}$$

$$\begin{aligned} \text{UYVY}[r][c/2+1] &= (\text{U}[r][c/4] \ll 24) \mid \\ &\quad (\text{Y}[r][c+2] \ll 16) \mid \\ &\quad (\text{V}[r][c/4] \ll 8) \mid \\ &\quad (\text{Y}[r][c+3]) \end{aligned}$$

where  $r = 0, 2, 4, \dots, h-2$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

*UYVY*     Pointer to output buffer.  
*Y*        Pointer to Y input buffer.  
*U*        Pointer to U input buffer.  
*V*        Pointer to V input buffer.  
*w*        Image width in pixels.  
*h*        Image height in lines.  
*dlb*      Linebytes for UYVY buffer.  
*ylb*      Linebytes for Y buffer.  
*uvlb*     Linebytes for U and V buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_UYVY422int\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV411seq\_to\_YUYV422int – convert YUV sequential to interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV411seq_to_YUYV422int(mllib_u32 *YUYV,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions *w* and *h*. The U and V buffers have dimensions *w*/4 and *h*. Dimension *w* is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The following equation is used:

$$\begin{aligned} \text{YUYV}[r][c/2] &= (\text{Y}[r][c] \ll 24) \mid \\ &\quad (\text{U}[r][c/4] \ll 16) \mid \\ &\quad (\text{Y}[r][c+1] \ll 8) \mid \\ &\quad (\text{V}[r][c/4]) \end{aligned}$$

$$\begin{aligned} \text{YUYV}[r][c/2+1] &= (\text{Y}[r][c+2] \ll 24) \mid \\ &\quad (\text{U}[r][c/4] \ll 16) \mid \\ &\quad (\text{Y}[r][c+3] \ll 8) \mid \\ &\quad (\text{V}[r][c/4]) \end{aligned}$$

where  $r = 0, 2, 4, \dots, h-2$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

|             |                                |
|-------------|--------------------------------|
| <i>YUYV</i> | Pointer to output buffer.      |
| <i>Y</i>    | Pointer to Y input buffer.     |
| <i>U</i>    | Pointer to U input buffer.     |
| <i>V</i>    | Pointer to V input buffer.     |
| <i>w</i>    | Image width in pixels.         |
| <i>h</i>    | Image height in lines.         |
| <i>dlb</i>  | Linebytes for YUYV buffer.     |
| <i>ylb</i>  | Linebytes for Y buffer.        |
| <i>uvlb</i> | Linebytes for U and V buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_UYVY422int\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoColorYUV420seq\_to\_ABGRint – color convert YUV sequential to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV420seq_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 const mllib_u8 *A_array, mllib_u8 A_const, mllib_s32 w, mllib_s32 h,
 mllib_s32 dlb, mllib_s32 yalb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions *w* and *h*. The U and V buffers have dimensions *w*/2 and *h*/2. Dimensions *w* and *h* are assumed to be even. Successive rows of the output buffer ABGR use successive rows of Y and the same rows of U and V.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |
| <i>uvlb</i>    | Linebytes for U and V buffers.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV420seq\_to\_ARGBint – color convert YUV sequential to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV420seq_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 yalb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/2$  and  $h/2$ . Dimensions  $w$  and  $h$  are assumed to be even. Successive rows of the output buffer ARGB use successive rows of Y and the same rows of U and V.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |
| <i>uvlb</i>    | Linebytes for U and V buffers.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV411seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV420seq\_to\_UYVY422int – convert YUV sequential to interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV420seq_to_UYVY422int(mllib_u32 *UYVY,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/2$  and  $h/2$ . Dimensions  $w$  and  $h$  are assumed to be even. Successive rows of the output buffer UYVY use successive rows of Y and the same rows of U and V.

The following equation is used:

$$\begin{aligned} \text{UYVY}[r][c/2] &= (\text{U}[r/2][c/2] \ll 24) \mid \\ &\quad (\text{Y}[r][c] \ll 16) \mid \\ &\quad (\text{V}[r/2][c/2] \ll 8) \mid \\ &\quad (\text{Y}[r][c+1]) \end{aligned}$$

$$\begin{aligned} \text{UYVY}[r+1][c/2] &= (\text{U}[r/2][c/2] \ll 24) \mid \\ &\quad (\text{Y}[r+1][c] \ll 16) \mid \\ &\quad (\text{V}[r/2][c/2] \ll 8) \mid \\ &\quad (\text{Y}[r+1][c+1]) \end{aligned}$$

where  $r = 0, 2, 4, \dots, h-2$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

|             |                                |
|-------------|--------------------------------|
| <i>UYVY</i> | Pointer to output buffer.      |
| <i>Y</i>    | Pointer to Y input buffer.     |
| <i>U</i>    | Pointer to U input buffer.     |
| <i>V</i>    | Pointer to V input buffer.     |
| <i>w</i>    | Image width in pixels.         |
| <i>h</i>    | Image height in lines.         |
| <i>dlb</i>  | Linebytes for UYVY buffer.     |
| <i>ylb</i>  | Linebytes for Y buffer.        |
| <i>uvlb</i> | Linebytes for U and V buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_UYVY422int\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV420seq\_to\_YUYV422int – convert YUV sequential to interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV420seq_to_YUYV422int(mllib_u32 *YUYV
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/2$  and  $h/2$ . Dimensions  $w$  and  $h$  are assumed to be even. Successive rows of the output buffer YUYV use successive rows of Y and the same rows of U and V.

The following equation is used:

$$\begin{aligned} \text{YUYV}[r][c/2] &= (\text{Y}[r][c] \ll 24) | \\ &\quad (\text{U}[r/2][c/2] \ll 16) | \\ &\quad (\text{Y}[r][c+1] \ll 8) | \\ &\quad (\text{V}[r/2][c/2]) \end{aligned}$$

$$\begin{aligned} \text{YUYV}[r+1][c/2] &= (\text{Y}[r+1][c] \ll 24) | \\ &\quad (\text{U}[r/2][c/2] \ll 16) | \\ &\quad (\text{Y}[r+1][c+1] \ll 8) | \\ &\quad (\text{V}[r/2][c/2]) \end{aligned}$$

where  $r = 0, 2, 4, \dots, h-2$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

|             |                                |
|-------------|--------------------------------|
| <i>YUYV</i> | Pointer to output buffer.      |
| <i>Y</i>    | Pointer to Y input buffer.     |
| <i>U</i>    | Pointer to U input buffer.     |
| <i>V</i>    | Pointer to V input buffer.     |
| <i>w</i>    | Image width in pixels.         |
| <i>h</i>    | Image height in lines.         |
| <i>dlb</i>  | Linebytes for YUYV buffer.     |
| <i>ylb</i>  | Linebytes for Y buffer.        |
| <i>uvlb</i> | Linebytes for U and V buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV411seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_UYVY422int\(3MLIB\)](#),[attributes\(5\)](#)



**Name** mllib\_VideoColorYUV422seq\_to\_ABGRint – color convert YUV sequential to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV422seq_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 yalb,
 mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions *w* and *h*. The U and V buffers have dimensions *w*/2 and *h*. Dimensions *w* and *h* are assumed to be even. Similar to `mllib_VideoColorYUV420seq_to_ABGRint()` except U and V are not sampled in the *h* direction.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |
| <i>uvlb</i>    | Linebytes for U and V buffers.     |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV420seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBint\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRint\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV422seq\_to\_ARGBint – color convert YUV sequential to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV422seq_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u8 *Y, const mllib_u8 *U,
 const mllib_u8 *V, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb,
 mllib_s32 yalb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions  $w$  and  $h$ . The U and V buffers have dimensions  $w/2$  and  $h$ . Dimensions  $w$  and  $h$  are assumed to be even. Similar to `mllib_VideoColorYUV420seq_to_ARGBint()` except U and V are not sampled in the  $h$  direction.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                                    |
|----------------|------------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.          |
| <i>Y</i>       | Pointer to Y input buffer.         |
| <i>U</i>       | Pointer to U input buffer.         |
| <i>V</i>       | Pointer to V input buffer.         |
| <i>A_array</i> | Array of alpha values.             |
| <i>A_const</i> | Constant alpha value.              |
| <i>w</i>       | Image width in pixels.             |
| <i>h</i>       | Image height in lines.             |
| <i>dlb</i>     | Linebytes for output buffer.       |
| <i>yalb</i>    | Linebytes for Y and alpha buffers. |

*uvlb*            Linebytes for U and V buffers.

**Return Values**   None.

**Attributes**    See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_VideoColorYUV420seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV420seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV411seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV422seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUYV422int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYVY422int\\_to\\_ABGRInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ARGBInt\(3MLIB\)](#),  
[mllib\\_VideoColorUYV444int\\_to\\_ABGRInt\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoColorYUV422seq\_to\_UYVY422int – convert YUV sequential to interleaved

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorYUV422seq_to_UYVY422int(mllib_u32 *UYVY,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions *w* and *h*. The U and V buffers have dimensions *w*/2 and *h*. Dimensions *w* and *h* are assumed to be even. Similar to `mllib_VideoColorYUV420seq_to_UYVY422int()` except U and V are not sampled in the *h* direction.

The following equation is used:

$$UYVY[r][c/2] = (U[r][c/2] \ll 24) \mid (Y[r][c] \ll 16) \mid (V[r][c/2] \ll 8) \mid (Y[r][c+1])$$

where *r* = 0, 1, 2, . . . , *h*-1; and *c* = 0, 2, 4, . . . , *w*-2.

**Parameters** The function takes the following arguments:

- UYVY*     Pointer to output buffer.
- Y*        Pointer to Y input buffer.
- U*        Pointer to U input buffer.
- V*        Pointer to V input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*      Linebytes for UYVY buffer.
- ylb*      Linebytes for Y buffer.
- uvlb*     Linebytes for U and V buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV411seq_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV422seq_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV420seq_to_UYVY422int(3MLIB),`  
`mllib_VideoColorYUV411seq_to_UYVY422int(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV422seq\_to\_YUYV422int – convert YUV sequential to interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUV422seq_to_YUYV422int(mllib_u32 *YUYV,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 ylb, mllib_s32 uvlb);
```

**Description** The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions *w* and *h*. The U and V buffers have dimensions *w*/2 and *h*. Dimensions *w* and *h* are assumed to be even. Similar to `mllib_VideoColorYUV420seq_to_YUYV422int()` except U and V are not sampled in the *h* direction.

The following equation is used:

$$\begin{aligned} \text{YUYV}[r][c/2] = & (\text{Y}[r][c] \ll 24) \mid \\ & (\text{U}[r][c/2] \ll 16) \mid \\ & (\text{Y}[r][c+1] \ll 8) \mid \\ & (\text{V}[r][c/2]) \end{aligned}$$

where *r* = 0, 1, 2, ..., *h*-1; and *c* = 0, 2, 4, ..., *w*-2.

**Parameters** The function takes the following arguments:

- YUYV*     Pointer to output buffer.
- Y*        Pointer to Y input buffer.
- U*        Pointer to U input buffer.
- V*        Pointer to V input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*      Linebytes for YUYV buffer.
- ylb*      Linebytes for Y buffer.
- uvlb*     Linebytes for U and V buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV411seq_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV420seq_to_UYVY422int(3MLIB),`  
`mllib_VideoColorYUV411seq_to_UYVY422int(3MLIB),`  
`mllib_VideoColorYUV422seq_to_UYVY422int(3MLIB),attributes(5)`



**Name** mllib\_VideoColorYUV444int\_to\_ABGRint – color convert YUV interleaved to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorYUV444int_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u8 *YUV, const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The YUV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the *Y* buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>YUV</i>     | Pointer to Y input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV444int\_to\_ARGBint – color convert YUV interleaved to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorYUV444int_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u8 *YUV, const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb, mllib_s32 alb);
```

**Description** The YUV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the *Y* buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.    |
| <i>YUV</i>     | Pointer to Y input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV444int\_to\_UYVY422int – convert YUV interleaved with subsampling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_VideoColorYUV444int_to_UYVY422int(mllib_u32 *UYVY,
 const mllib_u8 *YUV, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The YUV pixel stream is broken apart and recombined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The YUV buffer has dimensions  $w$  and  $h$ . Dimension  $w$  is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is Y[r][c], U[r][c], V[r][c], Y[r][c+1], U[r][c+1], V[r][c+1], ...

The following equation is used:

$$\begin{aligned} \text{UYVY}[r][c/2] = & (((U[r][c] + U[r][c+1]) / 2) \ll 24) | \\ & (Y[r][c] \ll 16) | \\ & (((V[r][c] + V[r][c+1]) / 2) \ll 8) | \\ & (Y[r][c+1]) \end{aligned}$$

where  $r = 0, 1, 2, \dots, h-1$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

**UYVY**     Pointer to output buffer.

**YUV**     Pointer to input buffer.

**w**        Image width in pixels.

**h**        Image height in lines.

**dlb**     Linebytes for output buffer.

**slb**     Linebytes for input buffer.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV444seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_UYVY422int\(3MLIB\)](#),

```
mllib_VideoColorUYV444int_to_YUYV422int(3MLIB),
mllib_VideoColorUYV444int_to_UYVY422int(3MLIB),attributes(5)
```

**Name** mllib\_VideoColorYUV444int\_to\_YUYV422int – convert YUV interleaved with subsampling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
void mllib_VideoColorYUV444int_to_YUYV422int(mllib_u32 *YUYV,
 const mllib_u8 *YUV, mllib_s32 w, mllib_s32 h, mllib_s32 dlb,
 mllib_s32 slb);
```

**Description** The YUV pixel stream is broken apart and recombined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The YUV buffer has dimensions  $w$  and  $h$ . Dimension  $w$  is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is Y[r][c], U[r][c], V[r][c], Y[r][c+1], U[r][c+1], V[r][c+1], ...

The following equation is used:

$$\begin{aligned} \text{YUYV}[r][c/2] = & (\text{Y}[r][c] \ll 24) \mid \\ & (((\text{U}[r][c] + \text{U}[r][c+1]) / 2) \ll 16) \mid \\ & (\text{Y}[r][c+1] \ll 8) \mid \\ & (((\text{V}[r][c] + \text{V}[r][c+1]) / 2)) \end{aligned}$$

where  $r = 0, 1, 2, \dots, h-1$ ; and  $c = 0, 2, 4, \dots, w-2$ .

**Parameters** The function takes the following arguments:

*YUYV*     Pointer to output buffer.  
*YUV*     Pointer to input buffer.  
*w*        Image width in pixels.  
*h*        Image height in lines.  
*dlb*     Linebytes for output buffer.  
*slb*     Linebytes for input buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoColorYUV444seq\\_to\\_YUYV422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444seq\\_to\\_UYVY422int\(3MLIB\)](#),  
[mllib\\_VideoColorYUV444int\\_to\\_UYVY422int\(3MLIB\)](#),

```
mllib_VideoColorUYV444int_to_YUYV422int(3MLIB),
mllib_VideoColorUYV444int_to_UYVY422int(3MLIB),attributes(5)
```



**Name** mllib\_VideoColorYUV444seq\_to\_ABGRint – color convert YUV sequential to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorYUV444seq_to_ABGRint(mllib_u32 *ABGR, const mllib_u8 *Y,
 const mllib_u8 *U, const mllib_u8 *V, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 slb);
```

**Description** The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>Y</i>       | Pointer to Y input buffer.   |
| <i>U</i>       | Pointer to U input buffer.   |
| <i>V</i>       | Pointer to V input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV444seq\_to\_ARGBint – color convert YUV sequential to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorYUV444seq_to_ARGBint(mllib_u32 *ARGB, const mllib_u8 *Y,
 const mllib_u8 *U, const mllib_u8 *V, const mllib_u8 *A_array,
 mllib_u8 A_const, mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 slb);
```

**Description** The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions *w* and *h*.

The alpha values for this function work in the following fashion:

- If *A\_array* pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If *A\_array* pointer is NULL, the alpha values for every pixel are set to *A\_const*.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} Y \\ U \\ V \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.    |
| <i>Y</i>       | Pointer to Y input buffer.   |
| <i>U</i>       | Pointer to U input buffer.   |
| <i>V</i>       | Pointer to V input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffers. |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV444seq\_to\_UYVY422int – convert YUV sequential to interleaved with subsampling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorYUV444seq_to_UYVY422int(mllib_u32 *UYVY,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb, mllib_s32 slb);
```

**Description** The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y, U, and V buffers have dimensions *w* and *h*. Dimension *w* is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values.

The following equation is used:

$$\begin{aligned} \text{UYVY}[r][c/2] = & (((U[r][c] + U[r][c+1]) / 2) \ll 24) | \\ & (Y[r][c] \ll 16) | \\ & (((V[r][c] + V[r][c+1]) / 2) \ll 8) | \\ & (Y[r][c+1]) \end{aligned}$$

where *r* = 0, 1, 2, . . . , *h*-1; and *c* = 0, 2, 4, . . . , *w*-2.

**Parameters** The function takes the following arguments:

- UYVY*     Pointer to output buffer.
- Y*        Pointer to Y input buffer.
- U*        Pointer to U input buffer.
- V*        Pointer to V input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*      Linebytes for output buffer.
- slb*      Linebytes for input buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV444seq_to_YUYV422int(3MLIB)`,  
`mllib_VideoColorYUV444int_to_YUYV422int(3MLIB)`,  
`mllib_VideoColorYUV444int_to_UYVY422int(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUV444seq\_to\_YUYV422int – convert YUV sequential to interleaved with subsampling

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

void mllib_VideoColorYUV444seq_to_YUYV422int(mllib_u32 *YUYV,
 const mllib_u8 *Y, const mllib_u8 *U, const mllib_u8 *V, mllib_s32 w,
 mllib_s32 h, mllib_s32 dlb, mllib_s32 slb);
```

**Description** The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y, U, and V buffers have dimensions *w* and *h*. Dimension *w* is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values.

The following equation is used:

$$YUYV[r][c/2] = (Y[r][c] \ll 24) |$$

$$(((U[r][c] + U[r][c+1]) / 2) \ll 16) |$$

$$(Y[r][c+1] \ll 8) |$$

$$(((V[r][c] + V[r][c+1]) / 2))$$

where *r* = 0, 1, 2, . . . , *h*-1; and *c* = 0, 2, 4, . . . , *w*-2.

**Parameters** The function takes the following arguments:

- YUYV*     Pointer to output buffer.
- Y*        Pointer to Y input buffer.
- U*        Pointer to U input buffer.
- V*        Pointer to V input buffer.
- w*        Image width in pixels.
- h*        Image height in lines.
- dlb*      Linebytes for output buffer.
- slb*      Linebytes for input buffers.

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV444seq_to_UYVY422int(3MLIB),`  
`mllib_VideoColorYUV444int_to_YUYV422int(3MLIB),`  
`mllib_VideoColorYUV444int_to_UYVY422int(3MLIB),attributes(5)`



**Name** mllib\_VideoColorYUYV422int\_to\_ABGRint – color convert YUYV interleaved to ABGR interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
void mllib_VideoColorYUYV422int_to_ABGRint(mllib_u32 *ABGR,
 const mllib_u32 *YUYV, const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** The YUYV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The YUYV buffer has dimensions  $w/2$  and  $h$ . The ABGR buffer has dimensions  $w$  and  $h$ . Dimensions  $w$  is assumed to be even.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ABGR</i>    | Pointer to output buffer.    |
| <i>YUYV</i>    | Pointer to Y input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoColorYUYV422int\_to\_ARGBint – color convert YUYV interleaved to ARGB interleaved

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
void mllib_VideoColorYUYV422int_to_ARGBint(mllib_u32 *ARGB,
 const mllib_u32 *YUYV, const mllib_u8 *A_array, mllib_u8 A_const,
 mllib_s32 w, mllib_s32 h, mllib_s32 dlb, mllib_s32 slb, mllib_s32 alb);
```

**Description** The YUYV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The YUYV buffer has dimensions  $w/2$  and  $h$ . The ABGR buffer has dimensions  $w$  and  $h$ . Dimensions  $w$  is assumed to be even.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} * \begin{bmatrix} |Y| \\ |U| \\ |V| \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}$$

**Parameters** The function takes the following arguments:

|                |                              |
|----------------|------------------------------|
| <i>ARGB</i>    | Pointer to output buffer.    |
| <i>YUYV</i>    | Pointer to Y input buffer.   |
| <i>A_array</i> | Array of alpha values.       |
| <i>A_const</i> | Constant alpha value.        |
| <i>w</i>       | Image width in pixels.       |
| <i>h</i>       | Image height in lines.       |
| <i>dlb</i>     | Linebytes for output buffer. |
| <i>slb</i>     | Linebytes for input buffer.  |
| <i>alb</i>     | Linebytes for alpha buffer.  |

**Return Values** None.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoColorYUV420seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUV420seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV411seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV422seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444seq_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorYUYV422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorYUV444int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYVY422int_to_ABGRint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ARGBint(3MLIB),`  
`mllib_VideoColorUYV444int_to_ABGRint(3MLIB),attributes(5)`

**Name** mllib\_VideoCopyRefAve\_U8\_U8\_16x16, mllib\_VideoCopyRefAve\_U8\_U8\_16x8, mllib\_VideoCopyRefAve\_U8\_U8\_8x16, mllib\_VideoCopyRefAve\_U8\_U8\_8x8, mllib\_VideoCopyRefAve\_U8\_U8\_8x4 – copies and averages a block from the reference block to the current block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoCopyRefAve_U8_U8_16x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRefAve_U8_U8_16x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRefAve_U8_U8_8x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRefAve_U8_U8_8x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRefAve_U8_U8_8x4(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);
```

**Description** Each of these functions copies and averages a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

**Parameters** Each of the functions takes the following arguments:

*curr\_block*      Pointer to the current block. *curr\_block* must be 8-byte aligned.

*ref\_block*        Pointer to the reference block.

*stride*            Stride, in bytes, between adjacent rows in both the current block and the reference block. *stride* must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#),

```
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),
mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),
mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_S16_U8_16x16(3MLIB),
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mlib\_VideoCopyRefAve\_U8\_U8 – copies and averages a block from the reference block to the current block

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VideoCopyRefAve_U8_U8(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
 mlib_s32 stride);
```

**Description** The `mlib_VideoCopyRefAve_U8_U8()` function copies and averages a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

*curr\_block*      Pointer to the current block. *curr\_block* must be 8-byte aligned.  
*ref\_block*        Pointer to the reference block.  
*width*            Width of the blocks  
*height*           Height of the blocks.  
*stride*           Stride, in bytes, between adjacent rows in both the current block and the reference block. stride must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpXY\\_S16\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpXY\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoInterpY\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpY\\_S16\\_U8\\_16x16\(3MLIB\)](#),

```
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```



**Name** mllib\_VideoCopyRef\_S16\_U8\_16x16, mllib\_VideoCopyRef\_S16\_U8\_16x8, mllib\_VideoCopyRef\_S16\_U8\_8x16, mllib\_VideoCopyRef\_S16\_U8\_8x8, mllib\_VideoCopyRef\_S16\_U8\_8x4 – copies a block from the reference block to the current block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoCopyRef_S16_U8_16x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_S16_U8_16x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_S16_U8_8x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_S16_U8_8x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_S16_U8_8x4(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 stride);
```

**Description** Each of these functions copies a block from the reference block to the motion-compensated reference block. The stride applies to only the input reference block.

**Parameters** Each of the functions takes the following arguments:

*mc\_block*      Pointer to the motion-compensated reference block. mc\_block must be 8-byte aligned.

*ref\_block*      Pointer to the reference block.

*stride*          Stride, in bytes, between adjacent rows in the reference block. stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#),

```
mllib_VideoInterpAveX_U8_U8_16x16(3MLIB),mllib_VideoInterpAveXY_U8_U8(3MLIB),
mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB),mllib_VideoInterpAveY_U8_U8(3MLIB),
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),
mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),
mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_S16_U8_16x16(3MLIB),
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoCopyRef\_S16\_U8 – copies a block from the reference block to the current block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoCopyRef_S16_U8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 stride);
```

**Description** The mllib\_VideoCopyRef\_S16\_U8() function copies a block from the reference block to the motion-compensated reference block. The stride applies to only the input reference block.

**Parameters** The function takes the following arguments:

- mc\_block* Pointer to the motion-compensated reference block. mc\_block must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- stride* Stride, in bytes, between adjacent rows in the reference block. stride must be a multiple of eight.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpY\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpY\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpY\\_U8\\_U8\\_16x16\(3MLIB\)](#),

```
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoCopyRef\_U8\_U8\_16x16, mllib\_VideoCopyRef\_U8\_U8\_16x8, mllib\_VideoCopyRef\_U8\_U8\_8x16, mllib\_VideoCopyRef\_U8\_U8\_8x8, mllib\_VideoCopyRef\_U8\_U8\_8x4 – copies an 8x8 block from the reference block to the current block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoCopyRef_U8_U8_16x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_U8_U8_16x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_U8_U8_8x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_U8_U8_8x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);

mllib_status mllib_VideoCopyRef_U8_U8_8x4(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);
```

**Description** Each of these functions copies a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

**Parameters** Each of the functions takes the following arguments:

*curr\_block*      Pointer to the current block. *curr\_block* must be 8-byte aligned.

*ref\_block*        Pointer to the reference block.

*stride*            Stride, in bytes, between adjacent rows in both the current block and the reference block. *stride* must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#),

```
mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB),mllib_VideoInterpAveY_U8_U8(3MLIB),
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),
mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),
mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_S16_U8_16x16(3MLIB),
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mlib\_VideoCopyRef\_U8\_U8 – copies a block from the reference block to the current block

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VideoCopyRef_U8_U8(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
 mlib_s32 stride);
```

**Description** The `mlib_VideoCopyRef_U8_U8()` function copies a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

*curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.

*ref\_block* Pointer to the reference block.

*width* Width of the blocks.

*height* Height of the blocks.

*stride* Stride, in bytes, between adjacent rows in both the current block and the reference block. *stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),  
[mlib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpXY\\_S16\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpXY\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoInterpY\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpY\\_S16\\_U8\\_16x16\(3MLIB\)](#),  
[mlib\\_VideoInterpY\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpY\\_U8\\_U8\\_16x16\(3MLIB\)](#),

```
mllib_VideoP64Decimate_U8_U8(3MLIB), mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
```



**Name** mlib\_VideoDCT16x16\_S16\_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_VideoDCT16x16_S16_S16(mlib_s16 coeffs[256],
 const mlib_s16 block[256]);
```

**Description** The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

**Parameters** The function takes the following arguments:

*coeffs* Pointer to the destination DCT coefficients. coeffs must be 8-byte aligned.

*block* Pointer to an 16x16 motion-compensated block that is the difference between the reference block and the current block. block must be 8-byte aligned.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mlib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mlib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mlib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mlib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mlib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT16x16\_S16\_S16\_B10 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT16x16_S16_S16_B10(mllib_s16 coeffs[256],
const mllib_s16 block[256]);
```

**Description** The mllib\_VideoDCT16x16\_S16\_S16\_B10() function computes the forward DCT for the destination DCT coefficients of data type mllib\_s16 and source block of data type mllib\_s16. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel which can occupy 10 bits, is represented as a 16-bit data.

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the output DCT coefficients. Note that coeffs must be 8-byte aligned.

*block*      Pointer to a 16x16 motion-compensated block which is the difference between the reference block and the current block. Note that block must be 8-byte aligned.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT2x2\_S16\_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT2x2_S16_S16(mllib_s16 coeffs[4],
 const mllib_s16 block[4]);
```

**Description** The `mllib_VideoDCT2x2_S16_S16()` function computes the forward DCT for the destination DCT coefficients of data type `mllib_s16` and a source block of data type `mllib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the destination DCT coefficients. *coeffs* must be 8-byte aligned.

*block*      Pointer to an 2x2 motion-compensated block that is the difference between the reference block and the current block. *block* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoDCT2x2_S16_S16(3MLIB)`, `mllib_VideoDCT4x4_S16_S16(3MLIB)`, `mllib_VideoDCT8x8_S16_S16(3MLIB)`, `mllib_VideoDCT8x8_S16_S16_B12(3MLIB)`, `mllib_VideoDCT8x8_S16_S16_NA(3MLIB)`, `mllib_VideoDCT8x8_S16_U8(3MLIB)`, `mllib_VideoDCT8x8_S16_U8_NA(3MLIB)`, `mllib_VideoDCT16x16_S16_S16(3MLIB)`, `mllib_VideoDCT16x16_S16_S16_B10(3MLIB)`, `mllib_VideoDeQuantize_S16(3MLIB)`, `mllib_VideoDeQuantizeInit_S16(3MLIB)`, `mllib_VideoQuantize_S16(3MLIB)`, `mllib_VideoQuantizeInit_S16(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_VideoDCT4x4\_S16\_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDCT4x4_S16_S16(mllib_s16 coeffs[16],
 const mllib_s16 block[16]);
```

**Description** The `mllib_VideoDCT4x4_S16_S16()` function computes the forward DCT for the destination DCT coefficients of data type `mllib_s16` and a source block of data type `mllib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

**Parameters** The function takes the following arguments:

- coeffs*     Pointer to the destination DCT coefficients. *coeffs* must be 8-byte aligned.
- block*      Pointer to an 4x4 motion-compensated block that is the difference between the reference block and the current block. *block* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoDCT2x2_S16_S16(3MLIB)`, `mllib_VideoDCT4x4_S16_S16(3MLIB)`, `mllib_VideoDCT8x8_S16_S16(3MLIB)`, `mllib_VideoDCT8x8_S16_S16_B12(3MLIB)`, `mllib_VideoDCT8x8_S16_S16_NA(3MLIB)`, `mllib_VideoDCT8x8_S16_U8(3MLIB)`, `mllib_VideoDCT8x8_S16_U8_NA(3MLIB)`, `mllib_VideoDCT16x16_S16_S16(3MLIB)`, `mllib_VideoDCT16x16_S16_S16_B10(3MLIB)`, `mllib_VideoDeQuantize_S16(3MLIB)`, `mllib_VideoDeQuantizeInit_S16(3MLIB)`, `mllib_VideoQuantize_S16(3MLIB)`, `mllib_VideoQuantizeInit_S16(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8Quantize\_S16\_S16\_B12 – forward Discrete Cosine Transform (DCT) and quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8Quantize_S16_S16_B12(
 mllib_s16 coeffs[64], const mllib_s16 block[64],
 const mllib_d64 dqtable[64]);
```

**Description** The `mllib_VideoDCT8x8Quantize_S16_S16_B12()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mllib_VideoDCT8x8_S16_S16_B12()` and `mllib_VideoQuantize_S16()` for better performance. The source to the DCT routine can occupy up to 12 bits for each of its elements, i.e., should be within the range of [-2048, 2047].

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters** The function takes the following arguments:

- coeffs* Pointer to the quantized DCT coefficients. Note that *coeffs* must be 8-byte aligned.
- block* Pointer to an 8x8 block. Note that *block* must be 8-byte aligned.
- dqtable* Pointer to the quantization table generated by `mllib_VideoQuantizeInit_S16()`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8Quantize\_S16\_S16\_B12\_NA – forward Discrete Cosine Transform (DCT) and quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8Quantize_S16_S16_B12_NA(
 mllib_s16 coeffs[64], const mllib_s16 block[64],
 const mllib_d64 dqtable[64]);
```

**Description** The `mllib_VideoDCT8x8Quantize_S16_S16_B12_NA()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mllib_VideoDCT8x8_S16_S16_B12_NA()` and `mllib_VideoQuantize_S16()` for better performance. The source to the DCT routine can occupy up to 12 bits for each of its elements, i.e., should be within the range of [-2048, 2047].

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters** The function takes the following arguments:

*coeffs*      Pointer to the quantized DCT coefficients.

*block*       Pointer to an 8x8 block.

*dqtable*     Pointer to the quantization table generated by `mllib_VideoQuantizeInit_S16()`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8Quantize\_S16\_U8 – forward Discrete Cosine Transform (DCT) and quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8Quantize_S16_U8(
 mllib_s16 coeffs[64], const mllib_u8 *block,
 const mllib_d64 dqtable[64], mllib_s32 stride);
```

**Description** The `mllib_VideoDCT8x8Quantize_S16_U8()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mllib_VideoDCT8x8_S16_U8()` and `mllib_VideoQuantize_S16()` for better performance.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

- coeffs* Pointer to the quantized DCT coefficients. Note that *coeffs* must be 8-byte aligned.
- block* Pointer to an 8x8 block. Note that *block* must be 8-byte aligned.
- dqtable* Pointer to the quantization table generated by `mllib_VideoQuantizeInit_S16()`.
- stride* Stride in bytes between adjacent rows in the block. Note that *stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8Quantize\_S16\_U8\_NA – forward Discrete Cosine Transform (DCT) and quantization

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8Quantize_S16_U8_NA(
 mllib_s16 coeffs[64], const mllib_u8 *block,
 const mllib_d64 dqtable[64], mllib_s32 stride);
```

**Description** The `mllib_VideoDCT8x8Quantize_S16_U8_NA()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mllib_VideoDCT8x8_S16_U8_NA()` and `mllib_VideoQuantize_S16()` for better performance.

This function requires no special address alignment.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

*coeffs*      Pointer to the quantized DCT coefficients.

*block*       Pointer to an 8x8 block.

*dqtable*     Pointer to the quantization table generated by `mllib_VideoQuantizeInit_S16()`.

*stride*       Stride in bytes between adjacent rows in the block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8Quantize\\_S16\\_U8\(3MLIB\)](#),[mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)



**Name** mlib\_VideoDCT8x8\_S16\_S16\_B10, mlib\_VideoDCT8x8\_S16\_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VideoDCT8x8_S16_S16_B10(
 mlib_s16 coeffs[64], const mlib_s16 block[64]);

mlib_status mlib_VideoDCT8x8_S16_S16(
 mlib_s16 coeffs[64], const mlib_s16 block[64]);
```

**Description** The `mlib_VideoDCT8x8_S16_S16_B10()` function computes the forward DCT for the destination DCT coefficients of data type `mlib_s16` and a source block of data type `mlib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum.

The source and destination buffer addresses must be 8-byte aligned.

Since mediaLib 2.5, `mlib_VideoDCT8x8_S16_S16()` has been renamed to `mlib_VideoDCT8x8_S16_S16_B10()`. Now `mlib_VideoDCT8x8_S16_S16()` is an alias of `mlib_VideoDCT8x8_S16_S16_B10()`.

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the destination DCT coefficients. *coeffs* must be 8-byte aligned.  
*block*     Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block. *block* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_S16\\_B10\\_NA\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mlib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mlib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8\_S16\_S16\_B10\_NA, mllib\_VideoDCT8x8\_S16\_S16\_NA – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8_S16_S16_B10_NA(
 mllib_s16 coeffs[64], const mllib_s16 block[64]);

mllib_status mllib_VideoDCT8x8_S16_S16_NA(
 mllib_s16 coeffs[64], const mllib_s16 block[64]);
```

**Description** The mllib\_VideoDCT8x8\_S16\_S16\_B10\_NA() function computes the forward DCT for the destination DCT coefficients of data type mllib\_s16 and a source block of data type mllib\_s16. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum.

This function requires no special address alignment.

Since mediaLib 2.5, mllib\_VideoDCT8x8\_S16\_S16\_NA() has been renamed to mllib\_VideoDCT8x8\_S16\_S16\_B10\_NA(). Now mllib\_VideoDCT8x8\_S16\_S16\_NA() is an alias of mllib\_VideoDCT8x8\_S16\_S16\_B10\_NA().

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the destination DCT coefficients.

*block*      Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8\_S16\_S16\_B12 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDCT8x8_S16_S16_B12(
 mllib_s16 coeffs[64], const mllib_s16 block[64]);
```

**Description** This function computes the forward DCT for the destination DCT coefficients of data type `mllib_s16` and source block of data type `mllib_s16`. The source to the DCT routine can occupy up to 12 bits for each of its elements.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision.

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the output DCT coefficients. Note that *coeffs* must be 8-byte aligned.

*block*     Pointer to an 8x8 block. Note that *block* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8\_S16\_U8 – forward Discrete Cosine Transform (DCT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoDCT8x8_S16_U8(mllib_s16 coeffs[64],
 const mllib_u8 *block, mllib_s32 stride);
```

**Description** The `mllib_VideoDCT8x8_S16_U8()` function computes the forward DCT for the destination DCT coefficients of data type `mllib_s16` and source block of data type `mllib_u8`. The stride applies to the block that is part of the frame currently being encoded.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

- coeffs*     Pointer to the destination DCT coefficients. Note that *coeffs* must be 8-byte aligned.
- block*     Pointer to an 8x8 block in the current frame. Note that *block* must be 8-byte aligned.
- stride*     Stride in bytes between adjacent rows in the block. Note that *stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#),  
[mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#),  
[mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDCT8x8\_S16\_U8\_NA – forward Discrete Cosine Transform (DCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDCT8x8_S16_U8_NA(mllib_s16 coeffs[64],
 const mllib_u8 *block, mllib_s32 stride);
```

**Description** The `mllib_VideoDCT8x8_S16_U8_NA()` function computes the forward DCT for the destination DCT coefficients of data type `mllib_s16` and source block of data type `mllib_u8`. The stride applies to the block that is part of the frame currently being encoded.

This function requires no special address alignment.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

*coeffs*     Pointer to the destination DCT coefficients.

*block*     Pointer to an 8x8 block in the current frame.

*stride*     Stride in bytes between adjacent rows in the block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B10\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDeQuantizeIDCT8x8\_S16\_S16\_B12 – dequantization and inverse Discrete Cosine Transform (IDCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12(
 mllib_s16 block[64], const mllib_s16 coeffs[64],
 const mllib_d64 dqtable[64]);
```

**Description** The `mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mllib_VideoDeQuantize_S16()` and `mllib_VideoIDCT8x8_S16_S16_B12()` for better performance. The output of this function should be within the range of `[-2048, 2047]` if `coeffs` is obtained from the corresponding `mllib_VideoDCT8x8Quantize_S16_S16_B12()` function.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters** The function takes the following arguments:

- block* Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.
- coeffs* Pointer to the input quantized DCT coefficients. Note that `coeffs` must be 8-byte aligned.
- dqtable* Pointer to the dequantization table generated by `mllib_VideoDeQuantizeInit_S16()`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDeQuantizeIDCT8x8\_S16\_S16\_B12\_NA – dequantization and inverse Discrete Cosine Transform (IDCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA(
 mllib_s16 block[64], const mllib_s16 coeffs[64],
 const mllib_d64 dqtable[64]);
```

**Description** The `mllib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mllib_VideoDeQuantize_S16()` and `mllib_VideoIDCT8x8_S16_S16_B12_NA()` for better performance. The output of this function should be within the range of  $[-2048, 2047]$  if `coeffs` is obtained from the corresponding `mllib_VideoDCT8x8Quantize_S16_S16_B12_NA()` function.

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters** The function takes the following arguments:

*block*        Pointer to an 8x8 block.

*coeffs*       Pointer to the input quantized DCT coefficients.

*dqtable*      Pointer to the dequantization table generated by `mllib_VideoDeQuantizeInit_S16()`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDeQuantizeIDCT8x8\_U8\_S16 – dequantization and inverse Discrete Cosine Transform (IDCT)

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDeQuantizeIDCT8x8_U8_S16(
 mllib_u8 *block, const mllib_s16 coeffs[64],
 const mllib_d64 dqtable[64], mllib_s32 stride);
```

**Description** The `mllib_VideoDeQuantizeIDCT8x8_U8_S16()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mllib_VideoDeQuantize_S16()` and `mllib_VideoIDCT8x8_U8_S16()` for better performance.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

- block* Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.
- coeffs* Pointer to the input quantized DCT coefficients. Note that `coeffs` must be 8-byte aligned.
- dqtable* Pointer to the dequantization table generated by `mllib_VideoDeQuantizeInit_S16()`.
- stride* Stride in bytes between adjacent rows in the block. Note that `stride` must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoDeQuantizeIDCT8x8\_U8\_S16\_NA – dequantization and inverse Discrete Cosine Transform (IDCT)

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoDeQuantizeIDCT8x8_U8_S16_NA(
 mllib_u8 *block, const mllib_s16 coeffs[64],
 const mllib_d64 dqtable[64], mllib_s32 stride);
```

**Description** The `mllib_VideoDeQuantizeIDCT8x8_U8_S16_NA()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mllib_VideoDeQuantize_S16()` and `mllib_VideoIDCT8x8_U8_S16_NA()` for better performance.

This function requires no special address alignment.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters** The function takes the following arguments:

- block* Pointer to an 8x8 block.
- coeffs* Pointer to the input quantized DCT coefficients.
- dqtable* Pointer to the dequantization table generated by `mllib_VideoDeQuantizeInit_S16()`.
- stride* Stride in bytes between adjacent rows in the block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_S16\\_S16\\_B12\\_NA\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeIDCT8x8\\_U8\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDeQuantizeInit\_S16 – dequantization of forward Discrete Cosine Transform (DCT) coefficients

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDeQuantizeInit_S16(mllib_d64 dqtable[64],
const mllib_s16 iqtable[64]);
```

**Description** The mllib\_VideoDeQuantizeInit\_S16() function initializes the dequantization table.

The following equation is used:

$$dqtable[i] = iqtable[i]; \quad 0 \leq i < 64$$

**Parameters** The function takes the following arguments:

- dqtable* Pointer to dequantizer table coefficients.
- iqtable* Pointer to original quantizer table coefficients:  
 $0 < iqtable[i] < 128$

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDeQuantize\_S16 – dequantization of forward Discrete Cosine Transform (DCT) coefficients

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDeQuantize_S16(mllib_s16 icoeffs[64],
 const mllib_d64 dqtable[64]);
```

**Description** The mllib\_VideoDeQuantize\_S16() function performs dequantization on DCT coefficients.

The following equation is used:

$$\text{icoeffs}[i] = \text{icoeffs}[i] * \text{dqtable}[i]; \quad 0 \leq i < 64$$

**Parameters** The function takes the following arguments:

*icoeffs*     Pointer to the output DCT coefficients:

$-2048 < \text{icoeffs}[i] < 2048$

Note that *icoeffs* must be 8-byte aligned.

*dqtable*     Pointer to dequantizer table coefficients.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDownSample420 – down sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDownSample420(mllib_u8 *dst, const mllib_u8 *src0,
 const mllib_u8 *src1, mllib_s32 n);
```

**Description** The `mllib_VideoDownSample420()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst*      Pointer to destination row. *dst* must be 8-byte aligned.
- src0*     Pointer to upper source row. *src0* must be 8-byte aligned.
- src1*     Pointer to middle source row. *src1* must be 8-byte aligned.
- n*        Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#),  
[mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#),  
[mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#),  
[mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#),  
[mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mlib\_VideoDownSample420\_S16 – down sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_VideoDownSample420_S16(mlib_s16 *dst,
 const mlib_s16 *src0, const mlib_s16 *src1, mlib_s32 n);
```

**Description** The `mlib_VideoDownSample420_S16()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*dst*      Pointer to destination row. *dst* must be 8-byte aligned.

*src0*     Pointer to upper source row. *src0* must be 8-byte aligned.

*src1*     Pointer to middle source row. *src1* must be 8-byte aligned.

*n*        Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoDownSample420\(3MLIB\)](#), [mlib\\_VideoDownSample422\(3MLIB\)](#), [mlib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample420\(3MLIB\)](#), [mlib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mlib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample422\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDownSample422 – down sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoDownSample422(mllib_u8 *dst, const mllib_u8 *src,
 mllib_s32 n);
```

**Description** The `mllib_VideoDownSample422()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst*     Pointer to destination row. *dst* must be 8-byte aligned.
- src*     Pointer to source row. *src* must be 8-byte aligned.
- n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoDownSample422\_S16 – down sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoDownSample422_S16(mllib_s16 *dst, const mllib_s16 *src,
 mllib_s32 n);
```

**Description** The `mllib_VideoDownSample422_S16()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination row. *dst* must be 8-byte aligned.

*src*     Pointer to source row. *src* must be 8-byte aligned.

*n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoH263OverlappedMC\_S16\_U8 – generates the 8x8 luminance prediction block in the Advanced Prediction Mode for H.263 codec

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoH263OverlappedMC_S16_U8(mllib_s16 mc_block[64],
 const mllib_u8 *ref_frame, mllib_s32 mch, mllib_s32 mcv, mllib_s32 mah,
 mllib_s32 mav, mllib_s32 mbh, mllib_s32 mbv, mllib_s32 mlh, mllib_s32 mlv,
 mllib_s32 mrh, mllib_s32 mrv, mllib_s32 ref_stride);
```

**Description** The `mllib_VideoH263OverlappedMC_S16_U8()` function generates an 8x8 luminance prediction block (motion-compensated block) in the Advanced Prediction Mode for H.263 codec. The reference frame in this function is an interpolated frame. The output of this function must be added to the IDCT output in order to reconstruct the block in the current frame.

The following equation is used:

for  $x = 0, 1, 2, 3; y = 0, 1, 2, 3$

$$mc(x, y) = (ref(2x + mch, 2y + mcv)*H0(x, y) + \\ ref(2x + mah, 2y + mav)*H1(x, y) + \\ ref(2x + mlh, 2y + mlv)*H2(x, y)) / 8;$$

for  $x = 4, 5, 6, 7; y = 0, 1, 2, 3$

$$mc(x, y) = (ref(2x + mch, 2y + mcv)*H0(x, y) + \\ ref(2x + mah, 2y + mav)*H1(x, y) + \\ ref(2x + mrh, 2y + mrv)*H2(x, y)) / 8;$$

for  $x = 0, 1, 2, 3; y = 4, 5, 6, 7$

$$mc(x, y) = (ref(2x + mch, 2y + mcv)*H0(x, y) + \\ ref(2x + mbh, 2y + mbv)*H1(x, y) + \\ ref(2x + mlh, 2y + mlv)*H2(x, y)) / 8;$$

for  $x = 4, 5, 6, 7; y = 4, 5, 6, 7$

$$mc(x, y) = (ref(2x + mch, 2y + mcv)*H0(x, y) + \\ ref(2x + mbh, 2y + mbv)*H1(x, y) + \\ ref(2x + mrh, 2y + mrv)*H2(x, y)) / 8;$$

where

$$H0 = \begin{bmatrix} 4 & 5 & 5 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \end{bmatrix}$$



```

 | 5 5 5 5 5 5 5 |
 [4 5 5 5 5 5 4]

 [2 2 2 2 2 2 2]
 | 1 1 2 2 2 2 1 |
 | 1 1 1 1 1 1 1 |
H1 = | 1 1 1 1 1 1 1 |
 | 1 1 1 1 1 1 1 |
 | 1 1 1 1 1 1 1 |
 | 1 1 2 2 2 2 1 |
 [2 2 2 2 2 2 2]

 [2 1 1 1 1 1 2]
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
H2 = | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 [2 1 1 1 1 1 2]

```

**Parameters** The function takes the following arguments:

|                  |                                                                                             |
|------------------|---------------------------------------------------------------------------------------------|
| <i>mc_block</i>  | Pointer to the motion-compensated block.                                                    |
| <i>ref_frame</i> | Pointer to the interpolated reference frame.                                                |
| <i>mch</i>       | Horizontal coordinate of the motion vector for the current block.                           |
| <i>mcv</i>       | Vertical coordinate of the motion vector for the current block.                             |
| <i>mah</i>       | Horizontal coordinate of the motion vector for the block above the current block.           |
| <i>mav</i>       | Vertical coordinate of the motion vector for the block above the current block.             |
| <i>mbh</i>       | Horizontal coordinate of the motion vector for the block below the current block.           |
| <i>mbv</i>       | Vertical coordinate of the motion vector for the block below the current block.             |
| <i>mlh</i>       | Horizontal coordinate of the motion vector for the block to the left of the current block.  |
| <i>mlv</i>       | Vertical coordinate of the motion vector for the block to the left of the current block.    |
| <i>mrh</i>       | Horizontal coordinate of the motion vector for the block to the right of the current block. |
| <i>mrv</i>       | Vertical coordinate of the motion vector for the block to the right of the current block.   |

*ref\_stride*      Stride, in bytes, between adjacent rows in the interpolated reference frame.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**      See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    `mllib_VideoAddBlock_U8_S16(3MLIB)`, `mllib_VideoCopyRef_S16_U8(3MLIB)`,  
`mllib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mllib_VideoCopyRef_U8_U8_16x16(3MLIB)`,  
`mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mllib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mllib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpX_S16_U8(3MLIB)`,  
`mllib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpX_U8_U8(3MLIB)`,  
`mllib_VideoInterpXY_S16_U8(3MLIB)`, `mllib_VideoInterpXY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpXY_U8_U8(3MLIB)`, `mllib_VideoInterpXY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_S16_U8(3MLIB)`, `mllib_VideoInterpY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_U8_U8(3MLIB)`, `mllib_VideoInterpY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoP64Decimate_U8_U8(3MLIB)`, `mllib_VideoP64Loop_S16_U8(3MLIB)`,  
`mllib_VideoP64Loop_U8_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoH263OverlappedMC\_U8\_U8 – generates the 8x8 luminance prediction block in the Advanced Prediction Mode for H.263 codec

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoH263OverlappedMC_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_frame, mllib_s32 mch, mllib_s32 mcv, mllib_s32 mah,
 mllib_s32 mav, mllib_s32 mbh, mllib_s32 mbv, mllib_s32 mlh, mllib_s32 mlv,
 mllib_s32 mrh, mllib_s32 mrv, mllib_s32 curr_stride,
 mllib_s32 ref_stride);
```

**Description** The `mllib_VideoH263OverlappedMC_U8_U8()` function generates an 8x8 luminance prediction block (motion-compensated block) in the Advanced Prediction Mode for H.263 codec. The reference frame in this function is an interpolated frame.

The following equation is used:

for  $x = 0, 1, 2, 3; y = 0, 1, 2, 3$

$$\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) * H0(x, y) + \\ \text{ref}(2x + \text{mah}, 2y + \text{mav}) * H1(x, y) + \\ \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) * H2(x, y)) / 8;$$

for  $x = 4, 5, 6, 7; y = 0, 1, 2, 3$

$$\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) * H0(x, y) + \\ \text{ref}(2x + \text{mah}, 2y + \text{mav}) * H1(x, y) + \\ \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) * H2(x, y)) / 8;$$

for  $x = 0, 1, 2, 3; y = 4, 5, 6, 7$

$$\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) * H0(x, y) + \\ \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) * H1(x, y) + \\ \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) * H2(x, y)) / 8;$$

for  $x = 4, 5, 6, 7; y = 4, 5, 6, 7$

$$\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) * H0(x, y) + \\ \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) * H1(x, y) + \\ \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) * H2(x, y)) / 8;$$

where

$$H0 = \begin{bmatrix} 4 & 5 & 5 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

```

 [4 5 5 5 5 5 4]

 [2 2 2 2 2 2 2]
 | 1 1 2 2 2 2 1 |
 | 1 1 1 1 1 1 1 |
H1 = | 1 1 1 1 1 1 1 |
 | 1 1 1 1 1 1 1 |
 | 1 1 1 1 1 1 1 |
 | 1 1 2 2 2 2 1 |
 [2 2 2 2 2 2 2]

 [2 1 1 1 1 1 2]
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
H2 = | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 | 2 2 1 1 1 1 2 |
 [2 1 1 1 1 1 2]

```

**Parameters** The function takes the following arguments:

|                   |                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------|
| <i>curr_block</i> | Pointer to the current block.                                                               |
| <i>ref_frame</i>  | Pointer to the interpolated reference frame.                                                |
| <i>mch</i>        | Horizontal coordinate of the motion vector for the current block.                           |
| <i>mcv</i>        | Vertical coordinate of the motion vector for the current block.                             |
| <i>mah</i>        | Horizontal coordinate of the motion vector for the block above the current block.           |
| <i>mav</i>        | Vertical coordinate of the motion vector for the block above the current block.             |
| <i>mbh</i>        | Horizontal coordinate of the motion vector for the block below the current block.           |
| <i>mbv</i>        | Vertical coordinate of the motion vector for the block below the current block.             |
| <i>mlh</i>        | Horizontal coordinate of the motion vector for the block to the left of the current block.  |
| <i>mlv</i>        | Vertical coordinate of the motion vector for the block to the left of the current block.    |
| <i>mrh</i>        | Horizontal coordinate of the motion vector for the block to the right of the current block. |
| <i>mrv</i>        | Vertical coordinate of the motion vector for the block to the right of the current block.   |

*curr\_stride*      Stride, in bytes, between adjacent rows in the current frame.

*ref\_stride*        Stride, in bytes, between adjacent rows in the interpolated reference frame.

**Return Values**    The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes**      See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also**    [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#),[mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#),[mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#),[mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#),  
[mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#),[mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#),  
[mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#),[mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#),  
[mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#),[mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#),[mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),  
[mllib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#),[mllib\\_VideoInterpXY\\_S16\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#),[mllib\\_VideoInterpXY\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoInterpY\\_S16\\_U8\(3MLIB\)](#),[mllib\\_VideoInterpY\\_S16\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoInterpY\\_U8\\_U8\(3MLIB\)](#),[mllib\\_VideoInterpY\\_U8\\_U8\\_16x16\(3MLIB\)](#),  
[mllib\\_VideoP64Decimate\\_U8\\_U8\(3MLIB\)](#),[mllib\\_VideoP64Loop\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoP64Loop\\_U8\\_U8\(3MLIB\)](#),[attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_S16\_S16\_B12, mllib\_VideoIDCT8x8\_S16\_S16 – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoIDCT8x8_S16_S16_B12(
 mllib_s16 block[64], const mllib_s16 coeffs[64]);

mllib_status mllib_VideoIDCT8x8_S16_S16(
 mllib_s16 block[64], const mllib_s16 coeffs[64]);
```

**Description** The `mllib_VideoIDCT8x8_S16_S16_B12()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mllib_s16` and input DCT coefficients of data type `mllib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine should be within the range of `[-2048, 2047]` if `coeffs` is obtained from the corresponding forward DCT function `mllib_VideoDCT8x8_S16_S16_B12()`.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision.

For MPEG, the output, which is really the difference between the current block and the reference block, can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

Since mediaLib 2.5, `mllib_VideoIDCT8x8_S16_S16()` has been renamed to `mllib_VideoIDCT8x8_S16_S16_B12()`. Now `mllib_VideoIDCT8x8_S16_S16()` is an alias of `mllib_VideoIDCT8x8_S16_S16_B12()`.

**Parameters** The function takes the following arguments:

- block*      Pointer to an 8x8 block in the current frame or motion-compensated reference block. *block* must be 8-byte aligned.
- coeffs*     Pointer to the source DCT coefficients. *coeffs* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoIDCT_IEEE_S16_S16(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_B12_NA(3MLIB)`,  
`mllib_VideoIDCT8x8_S16_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`,  
`mllib_VideoIDCT8x8_S16_S16_Q1_Mismatch(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16(3MLIB)`,  
`mllib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_NA(3MLIB)`,  
`mllib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoIDCT8x8\_S16\_S16\_B12\_NA, mllib\_VideoIDCT8x8\_S16\_S16\_NA – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoIDCT8x8_S16_S16_B12_NA(
 mllib_s16 block[64], const mllib_s16 coeffs[64]);

mllib_status mllib_VideoIDCT8x8_S16_S16_NA(
 mllib_s16 block[64], const mllib_s16 coeffs[64]);
```

**Description** The `mllib_VideoIDCT8x8_S16_S16_B12_NA()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mllib_s16` and input DCT coefficients of data type `mllib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine should be within the range of  $[-2048, 2047]$  if `coeffs` is obtained from the corresponding forward DCT function `mllib_VideoDCT8x8_S16_S16_B12_NA()`.

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision.

For MPEG, the output, which is really the difference between the current block and the reference block, can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

Since mediaLib 2.5, `mllib_VideoIDCT8x8_S16_S16_NA()` has been renamed to `mllib_VideoIDCT8x8_S16_S16_B12_NA()`. Now `mllib_VideoIDCT8x8_S16_S16_NA()` is an alias of `mllib_VideoIDCT8x8_S16_S16_B12_NA()`.

**Parameters** The function takes the following arguments:

- block*      Pointer to an 8x8 block in the current frame or motion-compensated reference block. `block` need not be 8-byte aligned.
- coeffs*      Pointer to the source DCT coefficients. `coeffs` need not be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** `mllib_VideoIDCT_IEEE_S16_S16(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_B12(3MLIB)`,  
`mllib_VideoIDCT8x8_S16_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`,  
`mllib_VideoIDCT8x8_S16_S16_Q1_Mismatch(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16(3MLIB)`,  
`mllib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_NA(3MLIB)`,  
`mllib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoIDCT8x8\_S16\_S16\_DC – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoIDCT8x8_S16_S16_DC(mllib_s16 *block,
 const mllib_s16 *coeffs);
```

**Description** The `mllib_VideoIDCT8x8_S16_S16_DC()` function can be used only when `F(0,0)` is nonzero. It computes the inverse DCT (called IDCT) for the output IDCT block of data type `mllib_s16` and input DCT coefficients of data type `mllib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

**Parameters** The function takes the following arguments:

- block*      Pointer to the current block. *block* must be 8-byte aligned.
- coeffs*     Pointer to the source DCT coefficients. *coeffs* must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT\\_IEEE\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_Q1\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_S16\_S16\_Q1 – inverse Discrete Cosine Transform

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoIDCT8x8_S16_S16_Q1(mllib_s16 block[64],
 const mllib_s16 coeffs[64]);
```

**Description** The `mllib_VideoIDCT8x8_S16_S16_Q1()` function can be used only when  $F(u, v)$  are nonzero for  $0 \leq u < 4$  and  $0 \leq v < 4$ .

**Parameters** The function takes the following arguments:

- block*      Pointer to the current block. block must be 8-byte aligned.
- coeffs*     Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoIDCT_IEEE_S16_S16(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_NA(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_NA(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_S16\_S16\_Q1\_Mismatch – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoIDCT8x8_S16_S16_Q1_Mismatch(
 mllib_s16 block[64], const mllib_s16 coeffs[64]);
```

**Description** The `mllib_VideoIDCT8x8_S16_S16_Q1_Mismatch()` function computes the inverse IDCT in the inter mode.

This function is similar to `mllib_VideoIDCT8x8_S16_S16_Q1()` which should only be used when `coeffs[u][v]` ( $u, v = 0 \dots 7$ ) are non-zero only for  $u$  and  $v$  less than 4. However, this function also allows element `coeffs[7][7]` to be non-zero. The primary benefit of this modification is that it can handle situations where `coeffs[7][7]` has been made non-zero by MPEG mismatch-control, allowing a simplified version of the IDCT to be undertaken for a much larger number of situations.

**Parameters** The function takes the following arguments:

- block* Pointer to an 8x8 motion-compensated block which is the difference between the reference block and current block. `block` must be 8-byte aligned.
- coeffs* Pointer to the input DCT coefficients. `coeffs` must be 8-byte aligned. `coeffs` should be in S12 range or it should be obtained from the corresponding forward DCT.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_U8\_S16 – inverse Discrete Cosine Transform

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>  
  
mllib_status mllib_VideoIDCT8x8_U8_S16(mllib_u8 *block,  
const mllib_s16 coeffs[64], mllib_s32 stride);`

**Description** The `mllib_VideoIDCT8x8_U8_S16()` function computes the inverse DCT (called IDCT) for the destination IDCT block of data type `mllib_u8` and source DCT coefficients of data type `mllib_s16`.

The stride applies to the block that is part of the frame currently being reconstructed.

**Parameters** The function takes the following arguments:

- block* Pointer to an 8x8 block in the current frame. block must be 8-byte aligned.
- coeffs* Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.
- stride* Stride, in bytes, between adjacent rows in a block. stride must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTETYPE       | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT\\_IEEE\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_Q1\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_U8\_S16\_DC – inverse Discrete Cosine Transform

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoIDCT8x8_U8_S16_DC(mllib_u8 *block,
 const mllib_s16 *coeffs, mllib_s32 stride);
```

**Description** The `mllib_VideoIDCT8x8_U8_S16_DC()` function can be used only when  $F(0,0)$  is nonzero. It computes the inverse DCT (called IDCT) for the destination IDCT block of data type `mllib_u8` and source DCT coefficients of data type `mllib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

**Parameters** The function takes the following arguments:

- block*      Pointer to the current block. *block* must be 8-byte aligned.
- coeffs*     Pointer to the source DCT coefficients. *coeffs* must be 8-byte aligned.
- stride*     Stride, in bytes, between adjacent rows in a block. *stride* must be a multiple of eight..

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT\\_IEEE\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_Q1\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_U8\_S16\_NA – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoIDCT8x8_U8_S16_NA(mllib_u8 *block,
 const mllib_s16 coeffs[64], mllib_s32 stride);
```

**Description** The `mllib_VideoIDCT8x8_U8_S16_NA()` function computes the inverse DCT (called IDCT) for the destination IDCT block of data type `mllib_u8` and source DCT coefficients of data type `mllib_s16`.

The `stride` applies to the block that is part of the frame currently being reconstructed.

This function requires no special address alignment.

**Parameters** The function takes the following arguments:

*block*      Pointer to the current block.

*coeffs*     Pointer to the source DCT coefficients.

*stride*     Stride, in bytes, between adjacent rows in the block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT\\_IEEE\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_Q1\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoIDCT8x8\_U8\_S16\_Q1 – inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoIDCT8x8_U8_S16_Q1(mllib_u8 *block,
 const mllib_s16 coeffs[64], mllib_s32 stride);
```

**Description** The `mllib_VideoIDCT8x8_U8_S16_Q1()` function can be used only when  $F(u, v)$  are nonzero and only when  $0 \leq u < 4$  and  $0 \leq v < 4$ . The stride applies to the block that is part of the frame currently being reconstructed.

**Parameters** The function takes the following arguments:

- block*      Pointer to the current block. block must be 8-byte aligned
- coeffs*     Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.
- stride*     Stride, in bytes, between adjacent rows in a block. stride must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoIDCT\\_IEEE\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_S16\\_S16\\_Q1\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_DC\(3MLIB\)](#), [mllib\\_VideoIDCT8x8\\_U8\\_S16\\_NA\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoIDCT\_IEEE\_S16\_S16 – IEEE-1180 compliant inverse Discrete Cosine Transform

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoIDCT_IEEE_S16_S16(mllib_s16 block[64],
 const mllib_s16 coeffs[64]);
```

**Description** The `mllib_VideoIDCT_IEEE_S16_S16()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mllib_s16` and input DCT coefficients of data type `mllib_s16`. This function is guaranteed to be IEEE-1180 -compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

**Parameters** The function takes the following arguments:

*block*      Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block. *block* need not be 8-byte aligned.

*coeffs*      Pointer to the source DCT coefficients. *coeffs* need not be 8-byte aligned.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoIDCT8x8_S16_S16(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_NA(3MLIB)`, `mllib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_NA(3MLIB)`, `mllib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`, [attributes\(5\)](#)

**Name** mllib\_VideoInterpAveX\_U8\_U8\_16x16, mllib\_VideoInterpAveX\_U8\_U8\_16x8, mllib\_VideoInterpAveX\_U8\_U8\_8x16, mllib\_VideoInterpAveX\_U8\_U8\_8x8, mllib\_VideoInterpAveX\_U8\_U8\_8x4 – half-pixel interpolation in the X direction and averaging for reference block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpAveX_U8_U8_16x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveX_U8_U8_16x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveX_U8_U8_8x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveX_U8_U8_8x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveX_U8_U8_8x4(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X direction and averaging for a reference block of data type mllib\_u8 and a current block of data type mllib\_u8. The stride applies to both the input reference block and the current block.

- Parameters** Each of the functions takes the following arguments:
- curr\_block* Pointer to the current block. curr\_block must be 8-byte aligned.
  - ref\_block* Pointer to the reference block.
  - frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm\_stride must be a multiple of eight.
  - fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld\_stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),`  
`mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),`  
`mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),`  
`mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),`  
`mlib_VideoH263OverlappedMC_S16_U8(3MLIB),`  
`mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),`  
`mlib_VideoInterpAveXY_U8_U8(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),`  
`mlib_VideoInterpAveY_U8_U8(3MLIB), mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),`  
`mlib_VideoInterpX_S16_U8(3MLIB), mlib_VideoInterpX_S16_U8_16x16(3MLIB),`  
`mlib_VideoInterpX_U8_U8(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),`  
`mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),`  
`mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),`  
`mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),`  
`mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),`  
`mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)`

**Name** mllib\_VideoInterpAveX\_U8\_U8 – half-pixel interpolation in the X direction and averaging for reference block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoInterpAveX_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpAveX_U8_U8()` function performs half-pixel interpolation in the X direction and averaging for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. The stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),

```
mlib_VideoInterpXY_S16_U8(3MLIB),mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB),mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB),mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB),mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoInterpAveXY\_U8\_U8\_16x16, mllib\_VideoInterpAveXY\_U8\_U8\_16x8, mllib\_VideoInterpAveXY\_U8\_U8\_8x16, mllib\_VideoInterpAveXY\_U8\_U8\_8x8, mllib\_VideoInterpAveXY\_U8\_U8\_8x4 – half-pixel interpolation in the X and Y directions and averaging for reference block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpAveXY_U8_U8_16x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride, mllib_s32 fld_stride);

mllib_status mllib_VideoInterpAveXY_U8_U8_16x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride, mllib_s32 fld_stride);

mllib_status mllib_VideoInterpAveXY_U8_U8_8x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride, mllib_s32 fld_stride);

mllib_status mllib_VideoInterpAveXY_U8_U8_8x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride, mllib_s32 fld_stride);

mllib_status mllib_VideoInterpAveXY_U8_U8_8x4(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X and Y directions and averaging for a reference block of data. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

- Parameters** Each of the functions takes the following arguments:
- curr\_block* Pointer to the current block. curr\_block must be 8-byte aligned.
  - ref\_block* Pointer to the reference block.
  - frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm\_stride must be a multiple of eight.
  - fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld\_stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`,  
`mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mlib_VideoCopyRef_U8_U8(3MLIB)`,  
`mlib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mlib_VideoCopyRefAve_U8_U8(3MLIB)`,  
`mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`,  
`mlib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mlib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mlib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mlib_VideoInterpAveY_U8_U8(3MLIB)`, `mlib_VideoInterpAveY_U8_U8_16x16(3MLIB)`,  
`mlib_VideoInterpX_S16_U8(3MLIB)`, `mlib_VideoInterpX_S16_U8_16x16(3MLIB)`,  
`mlib_VideoInterpX_U8_U8(3MLIB)`, `mlib_VideoInterpXY_S16_U8(3MLIB)`,  
`mlib_VideoInterpXY_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpXY_U8_U8(3MLIB)`,  
`mlib_VideoInterpXY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpY_S16_U8(3MLIB)`,  
`mlib_VideoInterpY_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpY_U8_U8(3MLIB)`,  
`mlib_VideoInterpY_U8_U8_16x16(3MLIB)`, `mlib_VideoP64Decimate_U8_U8(3MLIB)`,  
`mlib_VideoP64Loop_S16_U8(3MLIB)`, `mlib_VideoP64Loop_U8_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoInterpAveXY\_U8\_U8 – half-pixel interpolation in the X and Y directions and averaging for reference block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoInterpAveXY_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpAveXY_U8_U8()` function performs half-pixel interpolation in the X and Y directions and averaging for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),



```
mlib_VideoInterpX_S16_U8_16x16(3MLIB),mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB),mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB),mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB),mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB),mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoInterpAveY\_U8\_U8\_16x16, mllib\_VideoInterpAveY\_U8\_U8\_16x8, mllib\_VideoInterpAveY\_U8\_U8\_8x16, mllib\_VideoInterpAveY\_U8\_U8\_8x8, mllib\_VideoInterpAveY\_U8\_U8\_8x4 – half-pixel interpolation in the Y direction and averaging for reference block

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpAveY_U8_U8_16x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveY_U8_U8_16x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveY_U8_U8_8x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveY_U8_U8_8x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mllib_status mllib_VideoInterpAveY_U8_U8_8x4(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the Y direction and averaging for a reference block of data type mllib\_u8 and a current block of data type mllib\_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

- Parameters** Each of the functions takes the following arguments:
- curr\_block* Pointer to the current block. curr\_block must be 8-byte aligned.
  - ref\_block* Pointer to the reference block.
  - frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm\_stride must be a multiple of eight.
  - fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld\_stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB)`, `mllib_VideoCopyRef_S16_U8(3MLIB)`,  
`mllib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mllib_VideoCopyRef_U8_U8(3MLIB)`,  
`mllib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mllib_VideoCopyRefAve_U8_U8(3MLIB)`,  
`mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mllib_VideoH263OverlappedMC_S16_U8(3MLIB)`,  
`mllib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mllib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mllib_VideoInterpX_S16_U8(3MLIB)`, `mllib_VideoInterpX_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpX_U8_U8(3MLIB)`, `mllib_VideoInterpXY_S16_U8(3MLIB)`,  
`mllib_VideoInterpXY_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpY_S16_U8(3MLIB)`,  
`mllib_VideoInterpY_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpY_U8_U8(3MLIB)`,  
`mllib_VideoInterpY_U8_U8_16x16(3MLIB)`, `mllib_VideoP64Decimate_U8_U8(3MLIB)`,  
`mllib_VideoP64Loop_S16_U8(3MLIB)`, `mllib_VideoP64Loop_U8_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoInterpAveY\_U8\_U8 – half-pixel interpolation in the Y direction and averaging for reference block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoInterpAveY_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpAveY_U8_U8()` function performs half-pixel interpolation in the Y direction and averaging for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),

```
mlib_VideoInterpX_S16_U8_16x16(3MLIB),mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB),mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB),mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB),mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB),mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoInterpX\_S16\_U8\_16x16, mllib\_VideoInterpX\_S16\_U8\_16x8, mllib\_VideoInterpX\_S16\_U8\_8x16, mllib\_VideoInterpX\_S16\_U8\_8x8, mllib\_VideoInterpX\_S16\_U8\_8x4 – half-pixel interpolation in the X direction

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoInterpX_S16_U8_16x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_S16_U8_16x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_S16_U8_8x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_S16_U8_8x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_S16_U8_8x4(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X direction for a reference block of data type mllib\_u8 and a current block of data type mllib\_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** Each of the functions takes the following arguments:

|                   |                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>mc_block</i>   | Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.                                |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                    |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight. |

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),`  
`mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),`  
`mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),`  
`mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),`  
`mlib_VideoH263OverlappedMC_S16_U8(3MLIB),`  
`mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),`  
`mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),`  
`mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),`  
`mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),`  
`mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),`  
`mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),`  
`mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),`  
`mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),`  
`mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),`  
`mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)`

**Name** mllib\_VideoInterpX\_S16\_U8 – half-pixel interpolation in the X direction

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpX_S16_U8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpX_S16_U8()` function performs half-pixel interpolation in the X direction for a reference block of data type `mllib_u8` and a current block of data type `mllib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** The function takes the following arguments:

- mc\_block* Pointer to the motion-compensated reference block. *mc\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in the reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in the reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#),



```
mlib_VideoInterpX_S16_U8_16x16(3MLIB),mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB),mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB),mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB),mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB),mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoInterpX\_U8\_U8\_16x16, mllib\_VideoInterpX\_U8\_U8\_16x8, mllib\_VideoInterpX\_U8\_U8\_8x16, mllib\_VideoInterpX\_U8\_U8\_8x8, mllib\_VideoInterpX\_U8\_U8\_8x4 – half-pixel interpolation in the X direction

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoInterpX_U8_U8_16x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_U8_U8_16x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_U8_U8_8x16(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_U8_U8_8x8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpX_U8_U8_8x4(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X direction for a reference block of data type mllib\_u8 and a current block of data type mllib\_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** Each of the functions takes the following arguments:

- curr\_block*      Pointer to the current block. curr\_block must be 8-byte aligned.
- ref\_block*        Pointer to the reference block.
- frm\_stride*       Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. frm\_stride must be a multiple of eight.
- fld\_stride*       Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld\_stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB),mllib_VideoCopyRef_S16_U8(3MLIB),  
mllib_VideoCopyRef_S16_U8_16x16(3MLIB),mllib_VideoCopyRef_U8_U8(3MLIB),  
mllib_VideoCopyRef_U8_U8_16x16(3MLIB),mllib_VideoCopyRefAve_U8_U8(3MLIB),  
mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mllib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mllib_VideoH263OverlappedMC_U8_U8(3MLIB),mllib_VideoInterpAveX_U8_U8(3MLIB),  
mllib_VideoInterpAveX_U8_U8_16x16(3MLIB),mllib_VideoInterpAveXY_U8_U8(3MLIB),  
mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB),mllib_VideoInterpAveY_U8_U8(3MLIB),  
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),  
mllib_VideoInterpX_U8_U8(3MLIB),mllib_VideoInterpXY_S16_U8(3MLIB),  
mllib_VideoInterpXY_S16_U8_16x16(3MLIB),mllib_VideoInterpXY_U8_U8(3MLIB),  
mllib_VideoInterpXY_U8_U8_16x16(3MLIB),mllib_VideoInterpY_S16_U8(3MLIB),  
mllib_VideoInterpY_S16_U8_16x16(3MLIB),mllib_VideoInterpY_U8_U8(3MLIB),  
mllib_VideoInterpY_U8_U8_16x16(3MLIB),mllib_VideoP64Decimate_U8_U8(3MLIB),  
mllib_VideoP64Loop_S16_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)`

**Name** mllib\_VideoInterpX\_U8\_U8 – half-pixel interpolation in the X direction

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpX_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpX_U8_U8()` function performs half-pixel interpolation in the X direction for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#),

```
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
```

**Name** mllib\_VideoInterpXY\_S16\_U8\_16x16, mllib\_VideoInterpXY\_S16\_U8\_16x8, mllib\_VideoInterpXY\_S16\_U8\_8x16, mllib\_VideoInterpXY\_S16\_U8\_8x8, mllib\_VideoInterpXY\_S16\_U8\_8x4 – half-pixel interpolation in the X and Y directions for motion compensation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoInterpXY_S16_U8_16x16(mllib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_S16_U8_16x8(mllib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_S16_U8_8x16(mllib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_S16_U8_8x8(mllib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_S16_U8_8x4(mllib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X and Y directions for a reference block of data type mlib\_u8 and a current block of data type mlib\_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** Each of the functions takes the following arguments:

|                   |                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>mc_block</i>   | Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.                                |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                    |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight. |

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),`  
`mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),`  
`mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),`  
`mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),`  
`mlib_VideoH263OverlappedMC_S16_U8(3MLIB),`  
`mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),`  
`mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),`  
`mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),`  
`mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),`  
`mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),`  
`mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),`  
`mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),`  
`mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),`  
`mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),`  
`mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)`

**Name** mllib\_VideoInterpXY\_S16\_U8 – half-pixel interpolation in the X and Y directions

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpXY_S16_U8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpXY_S16_U8()` function performs half-pixel interpolation in the X and Y directions for a reference block of data type `mllib_u8` and a current block of data type `mllib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** The function takes the following arguments:

|                   |                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>mc_block</i>   | Pointer to the motion-compensated reference block. <i>mc_block</i> must be 8-byte aligned.                                |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                           |
| <i>width</i>      | Width of the blocks.                                                                                                      |
| <i>height</i>     | Height of the blocks.                                                                                                     |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in the reference block. <i>frm_stride</i> must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in the reference block. <i>fld_stride</i> must be a multiple of eight. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#),



```
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
```

**Name** mllib\_VideoInterpXY\_U8\_U8\_16x16, mllib\_VideoInterpXY\_U8\_U8\_16x8, mllib\_VideoInterpXY\_U8\_U8\_8x16, mllib\_VideoInterpXY\_U8\_U8\_8x8, mllib\_VideoInterpXY\_U8\_U8\_8x4 – half-pixel interpolation in the X and Y directions

**Synopsis** cc [ *flag...* ] *file...* -lmllib [ *library...* ]  
#include <mllib.h>

```
mllib_status mllib_VideoInterpXY_U8_U8_16x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mllib_s32 frm_stride,
mllib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_U8_U8_16x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mllib_s32 frm_stride,
mllib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_U8_U8_8x16(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mllib_s32 frm_stride,
mllib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_U8_U8_8x8(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mllib_s32 frm_stride,
mllib_s32 fld_stride);

mllib_status mllib_VideoInterpXY_U8_U8_8x4(mllib_u8 *curr_block,
const mllib_u8 *ref_block, mllib_s32 frm_stride,
mllib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the X and Y directions for a reference block of data type mllib\_u8 and a current block of data type mllib\_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** Each of the functions takes the following arguments:

- curr\_block* Pointer to the current block. curr\_block must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. frm\_stride must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld\_stride must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB),mllib_VideoCopyRef_S16_U8(3MLIB),  
mllib_VideoCopyRef_S16_U8_16x16(3MLIB),mllib_VideoCopyRef_U8_U8(3MLIB),  
mllib_VideoCopyRef_U8_U8_16x16(3MLIB),mllib_VideoCopyRefAve_U8_U8(3MLIB),  
mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mllib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mllib_VideoH263OverlappedMC_U8_U8(3MLIB),mllib_VideoInterpAveX_U8_U8(3MLIB),  
mllib_VideoInterpAveX_U8_U8_16x16(3MLIB),mllib_VideoInterpAveXY_U8_U8(3MLIB),  
mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB),mllib_VideoInterpAveY_U8_U8(3MLIB),  
mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),  
mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),  
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),  
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpY_S16_U8(3MLIB),  
mllib_VideoInterpY_S16_U8_16x16(3MLIB),mllib_VideoInterpY_U8_U8(3MLIB),  
mllib_VideoInterpY_U8_U8_16x16(3MLIB),mllib_VideoP64Decimate_U8_U8(3MLIB),  
mllib_VideoP64Loop_S16_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)`

**Name** mllib\_VideoInterpXY\_U8\_U8 – half-pixel interpolation in the X and Y directions

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpXY_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpXY_U8_U8()` function performs half-pixel interpolation in the X and Y directions for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.
- ref\_block* Pointer to the reference block.
- width* Width of the blocks.
- height* Height of the blocks.
- frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. *frm\_stride* must be a multiple of eight.
- fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),

```
mlib_VideoInterpXY_S16_U8(3MLIB),mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB),mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB),mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB),mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB),mlib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mllib\_VideoInterpX\_Y\_XY\_U8\_U8 – half-pixel interpolation in both X and Y directions for replenishment mode

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoInterpX_Y_XY_U8_U8(mllib_u8 *outputX,
 mllib_u8 *outputY, mllib_u8 *outputXY, const mllib_u8 *image,
 mllib_s32 stride, mllib_s32 width, mllib_s32 height);
```

**Description** The mllib\_VideoInterpX\_Y\_XY\_U8\_U8() function performs half-pixel interpolation in both X and Y directions for the replenishment mode.

**Parameters** The function takes the following arguments:

- outputX* Pointer to the output of X-interpolation. *outputX* must be 8-byte aligned.
- outputY* Pointer to the output of Y-interpolation. *outputY* must be 8-byte aligned.
- outputXY* Pointer to the output of XY-interpolation. *outputXY* must be 8-byte aligned.
- image* Pointer to the image data. *image* must be 8-byte aligned
- stride* Stride, in bytes, between adjacent rows in the image. *stride* must be a multiple of eight.
- width* Width of the image. *width* must be a multiple of eight.
- height* Height of the image. *height* must be a multiple of two.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpY\\_U8\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoInterpY\_S16\_U8\_16x16, mllib\_VideoInterpY\_S16\_U8\_16x8, mllib\_VideoInterpY\_S16\_U8\_8x16, mllib\_VideoInterpY\_S16\_U8\_8x8, mllib\_VideoInterpY\_S16\_U8\_8x4 – half-pixel interpolation in the Y direction

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoInterpY_S16_U8_16x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpY_S16_U8_16x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpY_S16_U8_8x16(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpY_S16_U8_8x8(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);

mllib_status mllib_VideoInterpY_S16_U8_8x4(mllib_s16 *mc_block,
 const mllib_u8 *ref_block, mllib_s32 frm_stride,
 mllib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the Y direction for a reference block of data type mllib\_u8 and a current block of data type mllib\_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** Each of the functions takes the following arguments:

|                   |                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>mc_block</i>   | Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.                                |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                    |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight. |

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB),mllib_VideoCopyRef_S16_U8(3MLIB),`  
`mllib_VideoCopyRef_S16_U8_16x16(3MLIB),mllib_VideoCopyRef_U8_U8(3MLIB),`  
`mllib_VideoCopyRef_U8_U8_16x16(3MLIB),mllib_VideoCopyRefAve_U8_U8(3MLIB),`  
`mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB),`  
`mllib_VideoH263OverlappedMC_S16_U8(3MLIB),`  
`mllib_VideoH263OverlappedMC_U8_U8(3MLIB),mllib_VideoInterpAveX_U8_U8(3MLIB),`  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB),mllib_VideoInterpAveXY_U8_U8(3MLIB),`  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB),mllib_VideoInterpAveY_U8_U8(3MLIB),`  
`mllib_VideoInterpAveY_U8_U8_16x16(3MLIB),mllib_VideoInterpX_S16_U8(3MLIB),`  
`mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),`  
`mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),`  
`mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),`  
`mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_U8_U8(3MLIB),`  
`mllib_VideoInterpY_U8_U8_16x16(3MLIB),mllib_VideoP64Decimate_U8_U8(3MLIB),`  
`mllib_VideoP64Loop_S16_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)`



**Name** mlib\_VideoInterpY\_S16\_U8 – half-pixel interpolation in the Y direction

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VideoInterpY_S16_U8(mlib_s16 *mc_block,
 const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
 mlib_s32 frm_stride, mlib_s32 fld_stride);
```

**Description** The `mlib_VideoInterpY_S16_U8()` function performs half-pixel interpolation in the Y direction for a reference block of data type `mlib_u8` and a current block of data type `mlib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters** The function takes the following arguments:

|                   |                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>mc_block</i>   | Pointer to the motion-compensated reference block. <i>mc_block</i> must be 8-byte aligned.                                |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                           |
| <i>width</i>      | Width of the blocks.                                                                                                      |
| <i>height</i>     | Height of the blocks.                                                                                                     |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in the reference block. <i>frm_stride</i> must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in the reference block. <i>fld_stride</i> must be a multiple of eight. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mlib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mlib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mlib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#),

```
mllib_VideoInterpX_S16_U8_16x16(3MLIB),mllib_VideoInterpX_U8_U8(3MLIB),
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),
mllib_VideoInterpY_S16_U8_16x16(3MLIB),mllib_VideoInterpY_U8_U8(3MLIB),
mllib_VideoInterpY_U8_U8_16x16(3MLIB),mllib_VideoP64Decimate_U8_U8(3MLIB),
mllib_VideoP64Loop_S16_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** mlib\_VideoInterpY\_U8\_U8\_16x16, mlib\_VideoInterpY\_U8\_U8\_16x8, mlib\_VideoInterpY\_U8\_U8\_8x16, mlib\_VideoInterpY\_U8\_U8\_8x8, mlib\_VideoInterpY\_U8\_U8\_8x4 – half-pixel interpolation in the Y direction

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]`  
`#include <mlib.h>`

```
mlib_status mlib_VideoInterpY_U8_U8_16x16(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

```
mlib_status mlib_VideoInterpY_U8_U8_16x8(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

```
mlib_status mlib_VideoInterpY_U8_U8_8x16(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

```
mlib_status mlib_VideoInterpY_U8_U8_8x8(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

```
mlib_status mlib_VideoInterpY_U8_U8_8x4(mlib_u8 *curr_block,
 const mlib_u8 *ref_block, mlib_s32 frm_stride,
 mlib_s32 fld_stride);
```

**Description** Each of these functions performs half-pixel interpolation in the Y direction for a reference block of data type mlib\_u8 and a current block of data type mlib\_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** Each of the functions takes the following arguments:

*curr\_block* Pointer to the current block. *curr\_block* must be 8-byte aligned.

*ref\_block* Pointer to the reference block.

*frm\_stride* Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. *frm\_stride* must be a multiple of eight.

*fld\_stride* Stride, in bytes, between adjacent rows in a field in both the current block and reference block. *fld\_stride* must be a multiple of eight.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level       | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB)`, `mllib_VideoCopyRef_S16_U8(3MLIB)`,  
`mllib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mllib_VideoCopyRef_U8_U8(3MLIB)`,  
`mllib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mllib_VideoCopyRefAve_U8_U8(3MLIB)`,  
`mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mllib_VideoH263OverlappedMC_S16_U8(3MLIB)`,  
`mllib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mllib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpX_S16_U8(3MLIB)`,  
`mllib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpX_U8_U8(3MLIB)`,  
`mllib_VideoInterpXY_S16_U8(3MLIB)`, `mllib_VideoInterpXY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpXY_U8_U8(3MLIB)`, `mllib_VideoInterpXY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_S16_U8(3MLIB)`, `mllib_VideoInterpY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_U8_U8(3MLIB)`, `mllib_VideoP64Decimate_U8_U8(3MLIB)`,  
`mllib_VideoP64Loop_S16_U8(3MLIB)`, `mllib_VideoP64Loop_U8_U8(3MLIB)`, `attributes(5)`

**Name** mllib\_VideoInterpY\_U8\_U8 – half-pixel interpolation in the Y direction

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoInterpY_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 frm_stride, mllib_s32 fld_stride);
```

**Description** The `mllib_VideoInterpY_U8_U8()` function performs half-pixel interpolation in the Y direction for a reference block of data type `mllib_u8` and a current block of data type `mllib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

|                   |                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>curr_block</i> | Pointer to the current block. <i>curr_block</i> must be 8-byte aligned.                                                                          |
| <i>ref_block</i>  | Pointer to the reference block.                                                                                                                  |
| <i>width</i>      | Width of the blocks.                                                                                                                             |
| <i>height</i>     | Height of the blocks.                                                                                                                            |
| <i>frm_stride</i> | Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. <i>frm_stride</i> must be a multiple of eight. |
| <i>fld_stride</i> | Stride, in bytes, between adjacent rows in a field in both the current block and reference block. <i>fld_stride</i> must be a multiple of eight. |

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH263OverlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#),

```
mllib_VideoInterpXY_S16_U8(3MLIB),mllib_VideoInterpXY_S16_U8_16x16(3MLIB),
mllib_VideoInterpXY_U8_U8(3MLIB),mllib_VideoInterpXY_U8_U8_16x16(3MLIB),
mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_S16_U8_16x16(3MLIB),
mllib_VideoInterpY_U8_U8_16x16(3MLIB),mllib_VideoP64Decimate_U8_U8(3MLIB),
mllib_VideoP64Loop_S16_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```

**Name** `mlib_VideoP64Decimate_U8_U8` – averages the source raster image over 2x2 blocks and writes the results to the destination raster image

**Synopsis** `cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>`

```
mlib_status mlib_VideoP64Decimate_U8_U8(mlib_u8 *dst,
 const mlib_u8 *src, mlib_s32 width, mlib_s32 height,
 mlib_s32 dst_stride, mlib_s32 src_stride);
```

**Description** The `mlib_VideoP64Decimate_U8_U8()` function averages the source raster image over 2x2 blocks and writes the results to the destination raster image. This function is used when the remote side is only capable of QCIF and our scanned image is source to the encoder in CIF format.

**Parameters** The function takes the following arguments:

*dst*            Pointer to the destination raster image.  
*src*            Pointer to the source raster image.  
*width*          Width of the image.  
*height*        Height of the image.  
*dst\_stride*     Stride, in bytes, between adjacent rows in the destination image.  
*src\_stride*     Stride, in bytes, between adjacent rows in the source image.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`,  
`mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mlib_VideoCopyRef_U8_U8_16x16(3MLIB)`,  
`mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mlib_VideoH2630verlappedMC_S16_U8(3MLIB)`,  
`mlib_VideoH2630verlappedMC_U8_U8(3MLIB)`, `mlib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mlib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mlib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpX_S16_U8(3MLIB)`,  
`mlib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpX_U8_U8(3MLIB)`,  
`mlib_VideoInterpXY_S16_U8(3MLIB)`, `mlib_VideoInterpXY_S16_U8_16x16(3MLIB)`,  
`mlib_VideoInterpXY_U8_U8(3MLIB)`, `mlib_VideoInterpXY_U8_U8_16x16(3MLIB)`,

```
mllib_VideoInterpY_S16_U8(3MLIB),mllib_VideoInterpY_S16_U8_16x16(3MLIB),
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
mllib_VideoP64Loop_U8_U8(3MLIB),attributes(5)
```



**Name** mllib\_VideoP64Loop\_S16\_U8 – applies a 2-dimensional (2D) 3x3 spatial filter on the reference block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoP64Loop_S16_U8(mllib_s16 mc_block[64],
 const mllib_u8 *ref_block, mllib_s32 stride);
```

**Description** The `mllib_VideoP64Loop_S16_U8()` function applies a 2-dimensional (2D) 3x3 spatial filter on the reference block. The filter is separable into 1D horizontal and vertical functions, where the filter coefficients are 0.25, 0.5, 0.25, except at the block edges where the coefficients are 0, 1, 0. In this mode, the output must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block. This function requires the motion-compensated block to be 8-bit aligned.

**Parameters** The function takes the following arguments:

- mc\_block*     Pointer to the motion-compensated reference block.
- ref\_block*     Pointer to the reference block.
- stride*        Stride, in bytes, between adjacent rows in the reference block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** `mllib_VideoAddBlock_U8_S16(3MLIB)`, `mllib_VideoCopyRef_S16_U8(3MLIB)`,  
`mllib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mllib_VideoCopyRef_U8_U8_16x16(3MLIB)`,  
`mllib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,  
`mllib_VideoH263OverlappedMC_S16_U8(3MLIB)`,  
`mllib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mllib_VideoInterpAveX_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveX_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveXY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpAveY_U8_U8(3MLIB)`,  
`mllib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mllib_VideoInterpX_S16_U8(3MLIB)`,  
`mllib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mllib_VideoInterpX_U8_U8(3MLIB)`,  
`mllib_VideoInterpXY_S16_U8(3MLIB)`, `mllib_VideoInterpXY_S16_U8_16x16(3MLIB)`,  
`mllib_VideoInterpXY_U8_U8(3MLIB)`, `mllib_VideoInterpXY_U8_U8_16x16(3MLIB)`,  
`mllib_VideoInterpY_S16_U8(3MLIB)`, `mllib_VideoInterpY_S16_U8_16x16(3MLIB)`,

```
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_U8_U8(3MLIB),
attributes(5)
```

**Name** mllib\_VideoP64Loop\_U8\_U8 – applies a 2-dimensional (2D) 3x3 spatial filter on the reference block

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoP64Loop_U8_U8(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 stride);
```

**Description** The `mllib_VideoP64Loop_U8_U8()` function applies a 2-dimensional (2D) 3x3 spatial filter on the reference block. The filter is separable into 1D horizontal and vertical functions, where the filter coefficients are 0.25, 0.5, 0.25, except at the block edges where the coefficients are 0, 1, 0. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:

- curr\_block* Pointer to the current block.
- ref\_block* Pointer to the reference block.
- stride* Stride, in bytes, between adjacent rows in both the current block and the reference block.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoAddBlock\\_U8\\_S16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRef\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoCopyRefAve\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoH2630verlappedMC\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveX\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpAveY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpX\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpX\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_S16\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_U8\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpXY\\_U8\\_U8\\_16x16\(3MLIB\)](#), [mllib\\_VideoInterpY\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoInterpY\\_S16\\_U8\\_16x16\(3MLIB\)](#),

```
mllib_VideoInterpY_U8_U8(3MLIB),mllib_VideoInterpY_U8_U8_16x16(3MLIB),
mllib_VideoP64Decimate_U8_U8(3MLIB),mllib_VideoP64Loop_S16_U8(3MLIB),
attributes(5)
```

**Name** mllib\_VideoQuantizeInit\_S16 – quantization of forward Discrete Cosine Transform (DCT) coefficients

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoQuantizeInit_S16(mllib_d64 dqtable[64],
 const mllib_s16 iqtable[64]);
```

**Description** The mllib\_VideoQuantizeInit\_S16() function initializes the quantization table.

The following equation is used:

$$dqtable[i] = 1.0 / iqtable[i]; \quad 0 \leq i < 64$$

**Parameters** The function takes the following arguments:

*dqtable*     Pointer to quantizer table coefficients.

*iqtable*     Pointer to original quantizer table coefficients:

$0 < iqtable[i] < 128$

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#), [mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantize\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoQuantize\_S16 – quantization of forward Discrete Cosine Transform (DCT) coefficients

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoQuantize_S16(mllib_s16 icoeffs[64],
 const mllib_d64 dqtable[64]);
```

**Description** The mllib\_VideoQuantize\_S16() function performs quantization on DCT coefficients.

The following equation is used:

$$\text{icoeffs}[i] = \text{icoeffs}[i] * \text{dqtable}[i]; \quad 0 \leq i < 64$$

**Parameters** The function takes the following arguments:

*icoeffs*     Pointer to the output DCT coefficients:  
              -2048 < icoeffs[i] < 2048  
              Note that icoeffs must be 8-byte aligned.

*dqtable*     Pointer to quantizer table coefficients.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDCT2x2\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT4x4\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8\\_S16\\_S16\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_S16\\_B12\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8\\_S16\\_S16\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT8x8\\_S16\\_U8\(3MLIB\)](#),  
[mllib\\_VideoDCT8x8\\_S16\\_U8\\_NA\(3MLIB\)](#), [mllib\\_VideoDCT16x16\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDCT16x16\\_S16\\_S16\\_B10\(3MLIB\)](#), [mllib\\_VideoDeQuantize\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDeQuantizeInit\\_S16\(3MLIB\)](#), [mllib\\_VideoQuantizeInit\\_S16\(3MLIB\)](#),  
[attributes\(5\)](#)

**Name** mllib\_VideoReversibleColorRGB2YUV\_U8\_U8,  
mllib\_VideoReversibleColorRGB2YUV\_S16\_U8,  
mllib\_VideoReversibleColorRGB2YUV\_S16\_S16,  
mllib\_VideoReversibleColorRGB2YUV\_S32\_S16 – reversible color space conversion for  
wavelet transformation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoReversibleColorRGB2YUV_U8_U8(mllib_u8 *y,
 mllib_u8 *u, mllib_u8 *v, const mllib_u8 *r, const mllib_u8 *g,
 const mllib_u8 *b, mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorRGB2YUV_S16_U8(mllib_s16 *y,
 mllib_s16 *u, mllib_s16 *v, const mllib_u8 *r, const mllib_u8 *g,
 const mllib_u8 *b,
 mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorRGB2YUV_S16_S16(mllib_s16 *y,
 mllib_s16 *u, mllib_s16 *v, const mllib_s16 *r, const mllib_s16 *g,
 const mllib_s16 *b, mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorRGB2YUV_S32_S16(mllib_s32 *y,
 mllib_s32 *u, mllib_s32 *v, const mllib_s16 *r, const mllib_s16 *g,
 const mllib_s16 *b, mllib_s32 n, mllib_s32 depth);
```

**Description** Each of the functions provides support to reversible wavelet transformation. It is for reversible color space conversion.

**Parameters** Each of the functions takes the following arguments:

|              |                                                                             |
|--------------|-----------------------------------------------------------------------------|
| <i>y</i>     | Pointer to destination Y component.                                         |
| <i>u</i>     | Pointer to destination U component.                                         |
| <i>v</i>     | Pointer to destination V component.                                         |
| <i>r</i>     | Pointer to source R component.                                              |
| <i>g</i>     | Pointer to source G component.                                              |
| <i>b</i>     | Pointer to source B component.                                              |
| <i>n</i>     | Length of data.                                                             |
| <i>depth</i> | Number of bit planes required to store the original R, G, and B components. |

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoReversibleColorYUV2RGB\\_U8\\_U8\(3MLIB\),attributes\(5\)](#)



**Name** mllib\_VideoReversibleColorYUV2RGB\_U8\_U8,  
 mllib\_VideoReversibleColorYUV2RGB\_U8\_S16,  
 mllib\_VideoReversibleColorYUV2RGB\_S16\_S16,  
 mllib\_VideoReversibleColorYUV2RGB\_S16\_S32 – reversible color space conversion for  
 wavelet transformation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoReversibleColorYUV2RGB_U8_U8(mllib_u8 *r,
 mllib_u8 *g, mllib_u8 *b, const mllib_u8 *y, const mllib_u8 *u,
 const mllib_u8 *v, mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorYUV2RGB_U8_S16(mllib_u8 *r,
 mllib_u8 *g, mllib_u8 *b, const mllib_s16 *y, const mllib_s16 *u,
 const mllib_s16 *v, mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorYUV2RGB_S16_S16(mllib_s16 *r,
 mllib_s16 *g, mllib_s16 *b, const mllib_s16 *y, const mllib_s16 *u,
 const mllib_s16 *v, mllib_s32 n, mllib_s32 depth);

mllib_status mllib_VideoReversibleColorYUV2RGB_S16_S32(mllib_s16 *r,
 mllib_s16 *g, mllib_s16 *b, const mllib_s32 *y, const mllib_s32 *u,
 const mllib_s32 *v, mllib_s32 n, mllib_s32 depth);
```

**Description** Each of the functions provides support to reversible wavelet transformation. It is for reversible color space conversion.

**Parameters** Each of the functions takes the following arguments:

*r*            Pointer to destination R component.  
*g*            Pointer to destination G component.  
*b*            Pointer to destination B component.  
*y*            Pointer to source Y component.  
*u*            Pointer to source U component.  
*v*            Pointer to source V component.  
*n*            Length of data.  
*depth*       Number of bit planes required to store the original R, G, and B components.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoReversibleColorRGB2YUV\\_U8\\_U8\(3MLIB\),attributes\(5\)](#)

**Name** mllib\_VideoSignMagnitudeConvert\_S16 – wavelet transformation, sign-magnitude conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoSignMagnitudeConvert_S16(mllib_s16 *srcdst,
mllib_s32 n);
```

**Description** The `mllib_VideoSignMagnitudeConvert_S16()` function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination data array.
- src* Pointer to source data array.
- srcdst* Pointer to source and destination data array.
- n* Array size.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoSignMagnitudeConvert\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\\_S32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoSignMagnitudeConvert\_S16\_S16 – wavelet transformation, sign-magnitude conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoSignMagnitudeConvert_S16_S16(mllib_s16 *dst,
const mllib_s16 *src, mllib_s32 n);
```

**Description** The mllib\_VideoSignMagnitudeConvert\_S16\_S16() function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

**Parameters** The function takes the following arguments:

- dst*        Pointer to destination data array.
- src*        Pointer to source data array.
- srcdst*    Pointer to source and destination data array.
- n*          Array size.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoSignMagnitudeConvert\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\\_S32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoSignMagnitudeConvert\_S32 – wavelet transformation, sign-magnitude conversion

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoSignMagnitudeConvert_S32(mllib_s32 *srcdst,
mllib_s32 n);
```

**Description** The `mllib_VideoSignMagnitudeConvert_S32()` function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination data array.
- src* Pointer to source data array.
- srcdst* Pointer to source and destination data array.
- n* Array size.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoSignMagnitudeConvert\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\\_S32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoSignMagnitudeConvert\_S32\_S32 – wavelet transformation, sign-magnitude conversion

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoSignMagnitudeConvert_S32_S32(mllib_s32 *dst,
const mllib_s32 *src, mllib_s32 n);
```

**Description** The `mllib_VideoSignMagnitudeConvert_S32_S32()` function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

**Parameters** The function takes the following arguments:

- dst* Pointer to destination data array.
- src* Pointer to source data array.
- srcdst* Pointer to source and destination data array.
- n* Array size.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoSignMagnitudeConvert\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S16\\_S16\(3MLIB\)](#),  
[mllib\\_VideoSignMagnitudeConvert\\_S32\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoSumAbsDiff – motion estimation

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_s32 mllib_VideoSumAbsDiff(mllib_u8 *curr_block,
 const mllib_u8 *ref_block, mllib_s32 width, mllib_s32 height,
 mllib_s32 stride);
```

**Description** The `mllib_VideoSumAbsDiff()` function computes the sum of absolute differences between the pixels in the current block and the corresponding pixels in the reference block.

Both the current block and the reference block belong to frames with the same dimension. (The stride is applicable to both.) Motion estimation computes the sum of the absolute differences between the current block and reference blocks at different locations in the reference frame, choosing the best fit (least sum of absolute difference) to calculate the motion vector.

**Parameters** The function takes the following arguments:

*curr\_block*      Pointer to the current block. *curr\_block* must be 8-byte aligned.

*ref\_block*        Pointer to the reference block.

*width*            Width of the block.

*height*           Height of the block.

*stride*           Stride, in bytes, between adjacent rows in a block. stride must be a multiple of eight.

**Return Values** The function returns a value of type `mllib_s32`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

**Name** mllib\_VideoUpSample420 – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoUpSample420(mllib_u8 *dst0, mllib_u8 *dst1,
 const mllib_u8 *src0, const mllib_u8 *src1,
 const mllib_u8 *src2, mllib_s32 n);
```

**Description** The mllib\_VideoUpSample420() function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst0* Pointer to upper destination row. dst0 must be 8-byte aligned.
- dst1* Pointer to lower destination row. dst1 must be 8-byte aligned.
- src0* Pointer to upper source row. src0 must be 8-byte aligned.
- src1* Pointer to middle source row. src1 must be 8-byte aligned.
- src2* Pointer to lower source row. src2 must be 8-byte aligned.
- n* Length of source rows. n must be greater than 1.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mlib\_VideoUpSample420\_Nearest – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmlib [library...]
#include <mlib.h>
```

```
mlib_status mlib_VideoUpSample420_Nearest(mlib_u8 *dst0, mlib_u8 *dst1,
 const mlib_u8 *src, mlib_s32 n);
```

**Description** The `mlib_VideoUpSample420_Nearest()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*dst0* Pointer to upper destination row. *dst0* must be 8-byte aligned.

*dst1* Pointer to lower destination row. *dst1* must be 8-byte aligned.

*src* Pointer to source row. *src* must be 8-byte aligned.

*n* Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mlib\\_VideoDownSample420\(3MLIB\)](#), [mlib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mlib\\_VideoDownSample422\(3MLIB\)](#), [mlib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample420\(3MLIB\)](#), [mlib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample422\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mlib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample420\_Nearest\_S16 – up sampling rate conversion in JFIF

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoUpSample420_Nearest_S16(mllib_s16 *dst0,
mllib_s16 *dst1, const mllib_s16 *src, mllib_s32 n);
```

**Description** The `mllib_VideoUpSample420_Nearest_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst0*     Pointer to upper destination row. *dst0* must be 8-byte aligned.
- dst1*     Pointer to lower destination row. *dst1* must be 8-byte aligned.
- src*       Pointer to source row. *src* must be 8-byte aligned.
- n*         Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#),  
[mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#),  
[mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#),  
[mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#),  
[mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample420\_S16 – up sampling rate conversion in JFIF

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VideoUpSample420_S16(mllib_s16 *dst0,
 mllib_s16 *dst1, const mllib_s16 *src0, const mllib_s16 *src1,
 const mllib_s16 *src2, mllib_s32 n);
```

**Description** The `mllib_VideoUpSample420_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst0*     Pointer to upper destination row. *dst0* must be 8-byte aligned.
- dst1*     Pointer to lower destination row. *dst1* must be 8-byte aligned.
- src0*     Pointer to upper source row. *src0* must be 8-byte aligned.
- src1*     Pointer to middle source row. *src1* must be 8-byte aligned.
- src2*     Pointer to lower source row. *src2* must be 8-byte aligned.
- n*        Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample422 – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VideoUpSample422(mllib_u8 *dst, const mllib_u8 *src,
 mllib_s32 n);
```

**Description** The `mllib_VideoUpSample422()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst*     Pointer to destination row. *dst* must be 8-byte aligned.
- src*     Pointer to source row. *src* must be 8-byte aligned.
- n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample422\_Nearest – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoUpSample422_Nearest(mllib_u8 *dst, const mllib_u8 *src,
 mllib_s32 n);
```

**Description** The `mllib_VideoUpSample422_Nearest()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

*dst*     Pointer to destination row. *dst* must be 8-byte aligned.

*src*     Pointer to source row. *src* must be 8-byte aligned.

*n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample422\_Nearest\_S16 – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoUpSample422_Nearest_S16(mllib_s16 *dst,
const mllib_s16 *src, mllib_s32 n);
```

**Description** The `mllib_VideoUpSample422_Nearest_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst*     Pointer to destination row. *dst* must be 8-byte aligned.
- src*     Pointer to source row. *src* must be 8-byte aligned.
- n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoUpSample422\_S16 – up sampling rate conversion in JFIF

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>
```

```
mllib_status mllib_VideoUpSample422_S16(mllib_s16 *dst,
 const mllib_s16 *src, mllib_s32 n);
```

**Description** The `mllib_VideoUpSample422_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters** The function takes the following arguments:

- dst*     Pointer to destination row. *dst* must be 8-byte aligned.
- src*     Pointer to source row. *src* must be 8-byte aligned.
- n*       Length of source rows. *n* must be greater than 1.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoDownSample420\(3MLIB\)](#), [mllib\\_VideoDownSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoDownSample422\(3MLIB\)](#), [mllib\\_VideoDownSample422\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_Nearest\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample420\\_S16\(3MLIB\)](#), [mllib\\_VideoUpSample422\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\(3MLIB\)](#), [mllib\\_VideoUpSample422\\_Nearest\\_S16\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VideoWaveletForwardTwoTenTrans,  
mllib\_VideoWaveletForwardTwoTenTrans\_S16\_U8,  
mllib\_VideoWaveletForwardTwoTenTrans\_S16\_S16,  
mllib\_VideoWaveletForwardTwoTenTrans\_S32\_S16,  
mllib\_VideoWaveletForwardTwoTenTrans\_S32\_S32 – wavelet transformation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoWaveletForwardTwoTenTrans_S16_U8(mllib_s16 *dst,
 const mlib_u8 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mllib_status mllib_VideoWaveletForwardTwoTenTrans_S16_S16(mllib_s16 *dst,
 const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mllib_status mllib_VideoWaveletForwardTwoTenTrans_S32_S16(mllib_s32 *dst,
 const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mllib_status mllib_VideoWaveletForwardTwoTenTrans_S32_S32(mllib_s32 *dst,
 const mlib_s32 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);
```

**Description** Each of the functions provides support to reversible wavelet transformation. It is for a forward two-ten transformation.

**Parameters** Each of the functions takes the following arguments:

- dst*            Pointer to TT-transform coefficients.
- src*            Pointer to source image.
- width*          Width of image.
- height*        Height of image.
- level*          Pointer to the number of decomposition levels. It returns the processed decomposition levels value.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoWaveletInverseTwoTenTrans\(3MLIB\)](#), [attributes\(5\)](#)



**Name** mllib\_VideoWaveletInverseTwoTenTrans,  
 mllib\_VideoWaveletInverseTwoTenTrans\_U8\_S16,  
 mllib\_VideoWaveletInverseTwoTenTrans\_S16\_S16,  
 mllib\_VideoWaveletInverseTwoTenTrans\_S16\_S32,  
 mllib\_VideoWaveletInverseTwoTenTrans\_S32\_S32 – wavelet transformation

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]`  
`#include <mllib.h>`

```
mllib_status mllib_VideoWaveletInverseTwoTenTrans_U8_S16(mllib_u8 *dst,
 const mllib_s16 *src, mllib_s32 width, mllib_s32 height, mllib_s32 *level);

mllib_status mllib_VideoWaveletInverseTwoTenTrans_S16_S16(mllib_s16 *dst,
 const mllib_s16 *src, mllib_s32 width, mllib_s32 height, mllib_s32 *level);

mllib_status mllib_VideoWaveletInverseTwoTenTrans_S16_S32(mllib_s16 *dst,
 const mllib_s32 *src, mllib_s32 width, mllib_s32 height, mllib_s32 *level);

mllib_status mllib_VideoWaveletInverseTwoTenTrans_S32_S32(mllib_s32 *dst,
 const mllib_s32 *src, mllib_s32 width, mllib_s32 height, mllib_s32 *level);
```

**Description** Each of the functions provides support to reversible wavelet transformation. It is for an inverse two-ten transformation.

**Parameters** Each of the functions takes the following arguments:

- dst*            Pointer to destination image.
- src*            Pointer to TT-transform coefficients.
- width*          Width of image.
- height*        Height of image.
- level*          Pointer to the number of decomposition levels. It returns the processed decomposition levels value.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VideoWaveletForwardTwoTenTrans\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeFindMaxBMask\_U8, mllib\_VolumeFindMaxBMask\_S16 – maximum intensity searching

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VolumeFindMaxBMask_U8(mllib_u8 *max,
 const mllib_rays *rays, const mllib_u8 *bmask);

mllib_status mllib_VolumeFindMaxBMask_S16(mllib_s16 *max,
 const mllib_rays *rays, const mllib_u8 *bmask);
```

**Description** Each function performs maximum intensity searching.

It uses the following equation:

```
max[i] = MAX{ rays->results[j][i]
 j = 0, 1, ..., rays->nsteps[i]; bmask[j] = 1 }
```

where  $i = 0, 1, \dots, \text{rays} \rightarrow \text{nrays} - 1$ .

**Parameters** The function takes the following arguments:

- max* Pointer to an array of `rays->nrays` maximum values of the samples in each ray.
- rays* Pointer to an `mllib_rays` structure. The data `rays->results` are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, `rays->results[rays->nsteps[i]][i]` on ray  $i$  might not equal 0.
- bmask* Pointer to a 1-bit mask array. Eight mask bits are packed into one byte. A 1 corresponds to the data in the step to be considered.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VolumeFindMax\\_U8\(3MLIB\)](#), [mllib\\_VolumeFindMaxCMask\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeFindMaxCMask\_U8, mllib\_VolumeFindMaxCMask\_S16 – maximum intensity searching

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VolumeFindMaxCMask_U8(mllib_u8 *max,
 const mllib_rays *rays, const mllib_u8 *cmask, mllib_s32 thresh);

mllib_status mllib_VolumeFindMaxCMask_S16(mllib_s16 *max,
 const mllib_rays *rays, const mllib_u8 *cmask, mllib_s32 thresh);
```

**Description** Each function performs maximum intensity searching.

It uses the following equation:

$$\max[i] = \text{MAX}\{ \text{rays} \rightarrow \text{results}[j][i] \mid j = 0, 1, \dots, \text{rays} \rightarrow \text{nsteps}[i]; \text{cmask}[j] > \text{thresh} \}$$

where  $i = 0, 1, \dots, \text{rays} \rightarrow \text{nrays} - 1$ .

**Parameters** The function takes the following arguments:

*max* Pointer to an array of `rays`→`nrays` maximum values of the samples in each ray.

*rays* Pointer to an `mllib_rays` structure. The data `rays`→`results` are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, `rays`→`results`[`rays`→`nsteps`[*i*]][*i*] on ray *i* might not equal 0.

*cmask* Pointer to an unsigned 8-bit mask array. If `cmask`[*j*] > `thresh`, then data in step *j*, `rays`→`results`[*j*], are considered.

*thresh* Threshold.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VolumeFindMax\\_U8\(3MLIB\)](#), [mllib\\_VolumeFindMaxBMask\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeFindMax\_U8, mllib\_VolumeFindMax\_S16 – maximum intensity searching

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VolumeFindMax_U8(mllib_u8 *max,
 const mllib_rays *rays);

mllib_status mllib_VolumeFindMax_S16(mllib_s16 *max,
 const mllib_rays *rays);
```

**Description** Each function performs maximum intensity searching.

It uses the following equation:

$$\text{max}[i] = \text{MAX}\{ \text{rays} \rightarrow \text{results}[j][i] \mid j = 0, 1, \dots, \text{rays} \rightarrow \text{nsteps}[i] \}$$

where  $i = 0, 1, \dots, \text{rays} \rightarrow \text{nrays} - 1$ .

**Parameters** The function takes the following arguments:

- max* Pointer to an array of `rays->nrays` maximum values of the samples in each ray.
- rays* Pointer to an `mllib_rays` structure. The data `rays->results` are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, `rays->results[rays->nsteps[i]][i]` on ray  $i$  might not equal 0.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VolumeFindMaxBMask\\_U8\(3MLIB\)](#), [mllib\\_VolumeFindMaxCMask\\_U8\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeRayCast\_Blocked, mllib\_VolumeRayCast\_Blocked\_Parallel\_Nearest\_U8\_U8, mllib\_VolumeRayCast\_Blocked\_Parallel\_Nearest\_S16\_S16, mllib\_VolumeRayCast\_Blocked\_Parallel\_Trilinear\_U8\_U8, mllib\_VolumeRayCast\_Blocked\_Parallel\_Trilinear\_S16\_S16, mllib\_VolumeRayCast\_Blocked\_Divergent\_Nearest\_U8\_U8, mllib\_VolumeRayCast\_Blocked\_Divergent\_Nearest\_S16\_S16, mllib\_VolumeRayCast\_Blocked\_Divergent\_Trilinear\_U8\_U8, mllib\_VolumeRayCast\_Blocked\_Divergent\_Trilinear\_S16\_S16 – cast a ray (or rays) through a 3D data set

**Synopsis**

```
cc [flag...] file... -lmllib [library...]
#include <mllib.h>

mllib_status mllib_VolumeRayCast_Blocked_Parallel_Nearest_U8_U8
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Parallel_Nearest_S16_S16
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Parallel_Trilinear_U8_U8
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Parallel_Trilinear_S16_S16
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Divergent_Nearest_U8_U8
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Divergent_Nearest_S16_S16
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Divergent_Trilinear_U8_U8
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);

mllib_status mllib_VolumeRayCast_Blocked_Divergent_Trilinear_S16_S16
 (mllib_rays *rays, const mllib_blkvolume *blk, void *buffer);
```

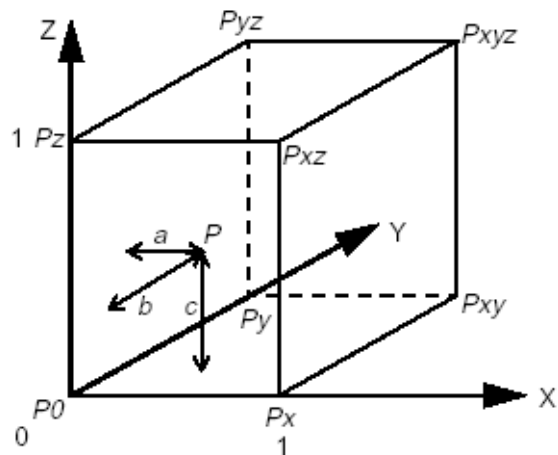
**Description** Each of these functions casts a ray (or rays) through a three-dimensional (3D) data set, then computes and returns the interpolated samples at each step along the way.

In trilinear interpolation, the value at point P is computed from its eight surrounding neighbors based on the equation below.

$$P = (1-a)*(1-b)*(1-c)*P0 + a*(1-b)*(1-c)*Px + (1-a)*b*(1-c)*Py + (1-a)*(1-b)*c*Pz + a*b*(1-c)*Pxy + a*(1-b)*c*Pxz + (1-a)*b*c*Pyz + a*b*c*Pxyz$$

where a, b, and c are the fractional parts of the coordinates of point P.

The trilinear interpolation is represented by the following figure:



In nearest neighbor operation, the sample value at point P is replaced by the value of the nearest neighbor voxel.

**Parameters** Each of the functions takes the following arguments:

- rays* Casting rays.
- blk* Volume data in the blocked format.
- buffer* Working buffer.

**Return Values** Each of the functions returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [mllib\\_VolumeRayCast\\_General\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeRayCast\_General, mllib\_VolumeRayCast\_General\_Parallel\_Nearest\_U8\_Bit, mllib\_VolumeRayCast\_General\_Parallel\_Nearest\_U8\_U8, mllib\_VolumeRayCast\_General\_Parallel\_Nearest\_S16\_S16, mllib\_VolumeRayCast\_General\_Parallel\_Trilinear\_U8\_U8, mllib\_VolumeRayCast\_General\_Parallel\_Trilinear\_S16\_S16, mllib\_VolumeRayCast\_General\_Divergent\_Nearest\_U8\_Bit, mllib\_VolumeRayCast\_General\_Divergent\_Nearest\_U8\_U8, mllib\_VolumeRayCast\_General\_Divergent\_Nearest\_S16\_S16, mllib\_VolumeRayCast\_General\_Divergent\_Trilinear\_U8\_U8, mllib\_VolumeRayCast\_General\_Divergent\_Trilinear\_S16\_S16 – cast a ray (or rays) through a 3D data set

**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VolumeRayCast_General_Parallel_Nearest_U8_Bit(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Parallel_Nearest_U8_U8(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Parallel_Nearest_S16_S16(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Parallel_Trilinear_U8_U8(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Parallel_Trilinear_S16_S16(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Divergent_Nearest_U8_Bit(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Divergent_Nearest_U8_U8(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Divergent_Nearest_S16_S16(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Divergent_Trilinear_U8_U8(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);

mllib_status mllib_VolumeRayCast_General_Divergent_Trilinear_S16_S16(
 mllib_rays *rays, const mllib_genvolume *vol, void *buffer);
```

**Description** Each of these functions casts a ray (or rays) through a three-dimensional (3D) data set, then computes and returns the interpolated samples at each step along the way.

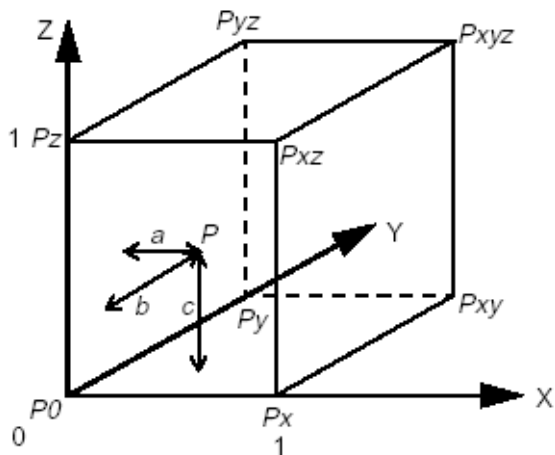
In trilinear interpolation, the value at point P is computed from its eight surrounding neighbors based on the equation below.

$$P = (1-a)*(1-b)*(1-c)*P_0 + a*(1-b)*(1-c)*P_x + (1-a)*b*(1-c)*P_y + (1-a)*(1-b)*c*P_z +$$

$$a*b*(1-c)*P_{xy} + a*(1-b)*c*P_{xz} + (1-a)*b*c*P_{yz} + a*b*c*P_{xyz}$$

where a, b, and c are the fractional parts of the coordinates of point P.

The trilinear interpolation is represented by the following figure:



In nearest neighbor operation, the sample value at point P is replaced by the value of the nearest neighbor voxel.

**Parameters** Each of the functions takes the following arguments:

- rays* Casting rays.
- vol* Volume data that consists of slices.
- buffer* Working buffer.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |



**See Also** [mlib\\_VolumeRayCast\\_Blocked\(3MLIB\)](#), [attributes\(5\)](#)

**Name** mllib\_VolumeWindowLevel – window-level operation

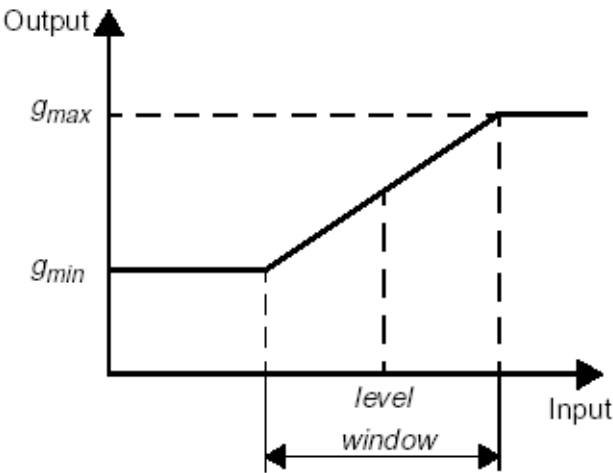
**Synopsis** `cc [ flag... ] file... -lmllib [ library... ]  
#include <mllib.h>`

```
mllib_status mllib_VolumeWindowLevel(mllib_u8 *dst,
 const mllib_s16 *src, mllib_s32 window, mllib_s32 level,
 mllib_s32 gmax, mllib_s32 gmin, mllib_s32 len);
```

**Description** The `mllib_VolumeWindowLevel()` function performs a window-level operation by using the following equation:

$$\begin{aligned} X &= x(i) \quad i = 0, 1, \dots, n-1 \\ Z &= z(i) \quad i = 0, 1, \dots, n-1 \\ &= \begin{cases} g_{min} & x(i) < l - \frac{w}{2} \\ (g_{max} - g_{min}) \frac{x(i) - (l - \frac{w}{2})}{w} + g_{min} & l - \frac{w}{2} \leq x(i) < l + \frac{w}{2} \\ g_{max} & x(i) \geq l + \frac{w}{2} \end{cases} \end{aligned}$$

The window-level operation is represented by the following figure:



**Parameters** The function takes the following arguments:

- dst* Pointer to the output or destination array.
- src* Pointer to the input or source array.
- window* Width of the window.
- level* Center of the window.
- gmax* Maximum grayscale in the destination array.
- gmin* Minimum grayscale in the destination array.
- len* Length of the data array.

**Return Values** The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

**Attributes** See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE      | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Committed       |
| MT-Level            | MT-Safe         |

**See Also** [attributes\(5\)](#)

